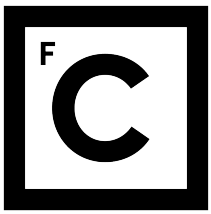
UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS



Ciências
ULisboa

Evolutionary Online Behaviour Learning and Adaptation in Robotic Systems

Doutoramento em Informática
Especialidade de Engenharia Informática

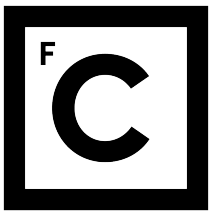
Fernando Goulart da Silva

Tese orientada por:
Prof. Doutor Luís Miguel Parreira e Correia (FC/UL)
Prof. Doutor Anders Lyhne Christensen (ISCTE-IUL)

Documento especialmente elaborado para a obtenção do grau de doutor

2017

UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS



Ciências
ULisboa

Evolutionary Online Behaviour Learning and Adaptation in Robotic Systems

Doutoramento em Informática
Especialidade de Engenharia Informática

Fernando Goulart da Silva

Tese orientada por:
Prof. Doutor Luís Miguel Parreira e Correia (FC/UL)
Prof. Doutor Anders Lyhne Christensen (ISCTE-IUL)

Júri

Presidente:

- Doutor Nuno Fuentecilla Maia Ferreira Neves

Vogais:

- Doutor Micael Santos Couceiro
- Doutor Leonardo Vanneschi
- Doutor José Nuno Panelas Nunes Lau
- Doutor Anders Lyhne Christensen (orientador)
- Doutora Sara Alexandra Cordeiro Madeira
- Doutor André Osório e Cruz de Azerêdo Falcão

Documento especialmente elaborado para a obtenção do grau de doutor

Instituição Financiadora: Fundação para a Ciência e Tecnologia (SFRH/BD/89573/2012)

2017

Acknowledgments

This journey has started in September 2007, when a 17-year-old version of me left Terceira Island, and moved to Lisbon. Since then, the naive, young boy has grown into a researcher, and is on his way to become a PhD. I owe much of it to my supervisors, Luís Correia and Anders Lyhne Christensen. Luís, you were the first to believe in me. You took the time to teach me, and were patient enough to withstand the reveries of a inexperienced student back in the cellular automata days. Without the chance you gave me, I would hardly be here. Anders, you have always been a caring mentor and a friend. You have helped me shape my way of thinking, and have lead by example with your commitment to excellence. After all these years, you are still able to surprise me with your vision of the world. Luís and Anders, I truly thank you for your mentorship, friendship, and patience to handle a wild thinker like me.

Being part of different research institutions has allowed me to come in contact with a number of different researchers. I acknowledge my friends and colleagues Davide Nunes, Jorge Gomes, Sancho Oliveira, and Vasco Costa, for their contribution to my personal and scientific growth. Special appreciation goes to Miguel Duarte and Nuno Henriques, two of the best people I have ever met. You are both tech savvies, great researchers, and engineers, yet it is on a personal level that you have really made a difference for me. Thank you for your friendship, advices, and willingness to discuss all sorts of personal and professional topics.

I would like to express my gratitude to my parents, José and Maria de Fátima, and to my brother André. You have believed in me, inspired me to always work hard, and encouraged me to pursue my passions. In that sense, this work is as much mine as it is yours. Without your teachings, my path would hardly have been the same.

Last, but without doubt on top, I would like to thank my fiancée Ana. I realise now that the doctoral process is much more than a scientific endeavour, it is a life experience in which personal aspects play a major role. No sentence will do justice to your love and support. Over the past years, you have nurtured me, encouraged me in the good times and in the bad times, and endured my long working hours and borderline-obsessed commitment to my work. We are on a roll, and I never want to stop. I love you.

This work was supported by Fundação para a Ciência e Tecnologia under grant SFRH/BD/89573/2012, project UID/MULTI/04046/2013, and project UID/EEA/50008/2013.

To my fiancée Ana, and to my parents José and Maria de Fátima.

Abstract

In this thesis, we study new ways to enable efficient online learning in autonomous robots. We employ a control synthesis methodology called *evolutionary robotics*, which emerged in the 1990s as a promising alternative to classic artificial intelligence techniques and design methodologies for control systems. In online learning through evolution, henceforth called *online evolution*, an evolutionary algorithm is executed onboard each robot in order to create and continuously optimise its behavioural control logic. Each instance of the evolutionary algorithm executes without any external supervision or human intervention. Online evolution can thus automatically generate the artificial intelligence that controls each robot, and creates the potential for long-term behaviour adaptation and learning: robots can continuously self-adjust and learn new behaviours in response to, for example, changes in the task requirements or environmental conditions, and to faults in the sensors and/or actuators.

Despite the potential for automatic behaviour learning, online evolution is not frequently employed for a number of reasons. First, online evolution typically requires several hours or days to synthesise solutions to a task. As a result, the approach has not yet been practically exploited in real-robot systems. Second, one common assumption in the field is that online evolution enables continuous learning and adaptation to previously unforeseen circumstances. However, only a small number of ad-hoc experiments have been carried out in simulation. That is, the potential for online evolution to enable dynamic adaptation and learning has been largely left unstudied. The main goal of this thesis is to address some of the fundamental issues associated with online evolution to bring it closer to widespread adoption.

Our research focuses on studying if and how to accelerate and increase the performance of online evolution. Our first research contribution is a comprehensive presentation and analysis of Online Decentralised NeuroEvolution of Augmenting Topologies (odNEAT), an algorithm for online evolution of neural network-based controllers in multirobot systems. odNEAT differs from more traditional approaches to online evolution because both the weighting parameters and the topological structure of neural networks are under evolutionary control. Our second research contribution focuses on investigating the dynamics of online evolution of controllers at two different levels. At the microscopic scale, we assess the dynamics of distinct neuronal models. At the macroscopic scale, we investigate the scalability properties of online evolution with respect to group size. The outcomes of the contribution are an assertion of the critical role of the controller evaluation policy, and an analysis of how the group size influences task performance. In our third research contribution, we capitalise on the knowledge gained from the second study, and we introduce: (i) a *racing* approach that allows individual robots to cut short the evaluation of poor controllers, (ii) a *population cloning* approach that enables each individual robot to clone and transmit a varying number of high-performing controllers to other robots nearby, and (iii) *online hyper-evolution* (OHE), an unprecedented approach in evolutionary robotics with the capability to automatically construct algorithms for controller generation during task execution. To conclude, we validate our research in real robotic hardware, and we successfully demonstrate evolution of controllers to solve three classic evolutionary robotics tasks in a timely manner (one hour or less).

Keywords: Online learning, online evolution, multirobot systems, artificial neural networks, adaptability.

Resumo Alargado

Nesta tese, estudamos novas abordagens para aprendizagem *online* em robôs autónomos. Fazemos uso de uma metodologia para síntese de controladores chamada *robótica evolucionária*, a qual emergiu na década de 1990 como uma alternativa promissora às técnicas clássicas de inteligência artificial e aos sistemas de controlo tradicionais. Na aprendizagem *online* através de métodos evolutivos, doravante designada evolução *online* (EO), um algoritmo evolucionário é executado por cada robô, de forma a criar e optimizar a sua lógica de controlo comportamental. Cada instância do algoritmo evolucionário executa sem supervisão externa ou intervenção humana. A EO pode assim gerar, de forma automática, a inteligência artificial que controla cada robô, e cria o potencial para aprendizagem e adaptação de comportamentos a longo prazo: os robôs podem auto-ajustar o seu comportamento e aprender novos comportamentos em resposta a, por exemplo, mudanças nos requisitos da tarefa ou nas condições ambientais, e a falhas nos sensores e/ou actuadores.

Apesar do potencial para aprendizagem automática e adaptação de comportamentos, a EO não é usada frequentemente por várias razões. Primeiro, a EO requer normalmente várias horas ou dias para sintetizar soluções para uma dada tarefa. Assim, a abordagem ainda não foi explorada de forma prática em robôs reais. Segundo, um pressuposto comum na robótica evolucionária é o de que a EO permite uma aprendizagem e adaptação contínuas quando os robôs se deparam com circunstâncias não experienciadas anteriormente. Todavia, apenas um pequeno número de experiências foram realizadas de forma a investigar este tópico. Para além disso, as experiências foram realizadas principalmente usando simulações, e de forma pouco sistematizada. Em suma, o potencial da EO em termos de adaptação e aprendizagem tem sido largamente deixado por investigar.

O principal objectivo desta tese é endereçar algumas das questões fundamentais associadas com a EO, para que esta se aproxime de um nível de maturidade em que possa ser adoptada de forma generalizada na robótica. Dado o potencial para aprendizagem colectiva, estudamos a EO em grupos de robôs autónomos. As nossas contribuições de investigação podem ser destiladas como se segue. Num primeiro estudo, avaliamos detalhadamente o algoritmo *Online Decentralised NeuroEvolution of Augmenting Topologies* (odNEAT), uma abordagem distribuída para a evolução em sistemas multi-robô de controladores baseados em redes neuronais. Os robôs evoluem controladores em paralelo, e transmitem uma cópia do seu controlador para robôs próximos. Contrariamente às abordagens anteriores, nas quais o experimentador humano tem de definir uma estrutura adequada para as redes neuronais, e apenas os paramêtros são optimizados evolucionariamente, o algoritmo odNEAT evolui tanto os parâmetros das redes neuronais como a sua estrutura. O algoritmo inicia a sua execução com redes minimais e complexifica-as de forma eficiente através da adição de novos neurónios e de novas ligações sinápticas. Desta forma, o odNEAT consegue encontrar, de forma automática, uma estrutura topológica para as redes neuronais adequada à tarefa que se pretende que os robôs resolvam.

Num segundo conjunto de estudos, investigamos as dinâmicas da EO a dois níveis diferentes. A um nível microscópico, investigamos a dinâmica de controladores baseados em redes neuronais com diferentes modelos de neurónios, nomeadamente *summing neurons*, *multiplicative neurons*, e uma combinação de ambos. O estudo é motivado por avanços na área de aprendizagem automática, onde os *multiplicative neurons* têm demonstrado aumentar a capacidade computacional das redes neuronais. A um nível macroscópico, investigamos as propriedades de escalabilidade da EO em relação ao tamanho do grupo de robôs. Quando o algoritmo evolucionário é distribuído por um grupo de robôs, existe o pressuposto comum de que a EO é inerentemente escalável. De

uma forma geral, o pressuposto consiste na ideia de que quantos mais robôs estiverem disponíveis, maior o número de avaliações de controladores que poderão ser realizadas em paralelo, e mais rápido será o processo evolucionário. A própria dinâmica do algoritmo evolucionário, bem como questões relacionadas com o dimensionamento da população, como taxas de convergência e diversidade, não são consideradas. Para além disso, não existe qualquer outro estudo sistemático sobre as propriedades de escalabilidade de algoritmos de EO em diferentes tarefas, o que motiva a nossa investigação.

Em seguida, apresentamos dois casos de estudo. No primeiro caso de estudo, introduzimos duas novas abordagens para acelerar a EO de controladores em sistemas multi-robô, nomeadamente: (i) uma abordagem de *racing* que permite a cada um dos robôs abortar a avaliação de controladores com baixo desempenho, e (ii) uma abordagem de clonagem de população, que permite aos robôs clonar um conjunto de controladores com um nível de desempenho relativamente alto, e transmitir esses clones para outros robôs na vizinhança. A motivação subjacente à técnica de clonagem de população é capitalizar, com vista a um melhor desempenho, a informação genética acumulada por vários robôs ao longo do processo evolucionário. No segundo caso de estudo, introduzimos a *online hyper-evolution* (OHE), uma abordagem em que o princípio-chave é utilizar as diferentes fontes de informação tradicionalmente associadas com a avaliação de controladores, nomeadamente desempenho na tarefa (*fitness*) e comportamento (novidade ou diversidade), de forma a seleccionar e/ou construir algoritmos para EO. Contrariamente às abordagens actuais, que se baseiam num único algoritmo, que é predefinido e não se altera durante a execução das tarefas, para encontrar um controlador de alto desempenho, a OHE procura num espaço de algoritmos, operadores evolucionários, e componentes algoritmicos (ex: mutação, recombinação, entre outros), para encontrar algoritmos eficazes e eficientes ao longo do tempo. A capacidade de construir automaticamente algoritmos para a geração de controladores não tem precedente na EO, e introduz um novo paradigma: permitir aos robôs melhorar o seu próprio processo de aprendizagem à medida que executam.

No último conjunto de estudos, reportamos a evolução de controladores em robôs reais, com vista à resolução de três tarefas clássicas na robótica evolucionária. Para as três tarefas, as soluções são evoluídas a partir de controladores gerados aleatoriamente ou de controladores pré-evoluídos em simulação. Em todos os casos, controladores capazes de resolver a tarefa são sintetizados num intervalo de tempo reduzido (uma hora ou menos). Demonstramos ainda, pela primeira vez, as capacidades adaptativas da EO em robôs reais, incluindo um grupo de robôs capaz de superar falhas injectadas em simultâneo nos motores de múltiplos robôs. Para finalizar, demonstramos que o desempenho e capacidade de adaptação dos robôs reais é facilitada pelo algoritmo evolucionário subjacente, odNEAT.

Em geral, os nossos desenvolvimentos constituem um passo em frente na EO, na medida em que: (i) permitem diferentes melhoramentos em termos, por exemplo, de desempenho e tempo necessário para sintetizar soluções para tarefas, e (ii) demonstram o potencial da EO para permitir uma aprendizagem e adaptação contínuas. Para além disso, esta tese apresenta formas de pensar a EO radicalmente diferentes do que foi estudado até agora, como a OHE, e deixa o caminho aberto a novas abordagens e tópicos de investigação.

Palavras-chave: Aprendizagem *online*, evolução *online*, sistemas multi-robô, redes neuronais, adaptabilidade.

Contents

List of Figures **xvii**

List of Tables **xix**

1 Introduction **1**

1.1 Problem Statement . . . 4
1.2 Thesis Structure and Contribution of Research . . . 5
1.3 Other Scientific Contributions . . . 8
1.4 Software Tools . . . 9
1.4.1 Simulation Platform . . . 9
1.4.2 Common Interface, Raspberry Controller, and Robot Control Console . . . 10
1.5 Summary . . . 10

2 Background **13**

2.1 Introduction to Evolutionary Robotics . . . 13
2.1.1 Basic Framework . . . 13
2.1.2 Guiding the Search in Evolutionary Robotics . . . 14
2.1.3 Early Pioneering Contributions . . . 18
2.1.4 Open Issues in Evolutionary Robot Engineering . . . 19
2.2 Online Evolution of Behavioural Control . . . 20
2.2.1 Classifying the Online Evolutionary Process . . . 21
2.2.2 Background on Online Evolution of Control . . . 21
2.3 Artificial Neural Networks . . . 27
2.3.1 Neurons and Activation Functions . . . 28
2.3.2 Artificial Neural Network Architectures . . . 28
2.4 Neuroevolution . . . 30
2.4.1 Fixed-Topology Neuroevolution Algorithms . . . 30
2.4.2 Constructive Neuroevolution Algorithms . . . 31
2.5 Summary . . . 35

3 Online Decentralised NeuroEvolution of Augmenting Topologies **37**

3.1 IM-($\mu + 1$) algorithm . . . 37

3.1.1 Execution Loop . . . 38
3.2 Introducing odNEAT . . . 38
3.2.1 Virtual Energy Level and Fitness Score . . . 39
3.2.2 Internal Population, Exchange of Genomes, and Tabu List . . . 40
3.2.3 New Genomes and the Maturation Period . . . 40
3.3 Experimental Methodology . . . 41
3.3.1 Experimental Setup . . . 42
3.3.2 Preliminary Performance Tuning . . . 42
3.3.3 Aggregation . . . 43
3.3.4 Integrated Navigation and Obstacle Avoidance . . . 44
3.3.5 Phototaxis . . . 45
3.3.6 Treatments . . . 46
3.4 Experimental Results . . . 46
3.4.1 Comparing odNEAT and rtNEAT . . . 46
3.4.2 Comparing odNEAT and IM-($\mu + 1$) . . . 50
3.5 Assessing odNEAT: Fault Injection Experiments and Ablation Studies . . . 52
3.5.1 Adaptation Performance . . . 52
3.5.2 Ablation Studies . . . 55
3.6 Summary . . . 57

4 Dynamics of Online Evolution of Robotic Controllers 59
4.1 Neuronal Model . . . 60
4.1.1 Experimental Methodology . . . 60
4.1.2 Experimental Results . . . 62
4.2 Scalability in Groups of Robots . . . 66
4.2.1 Experimental Methodology . . . 66
4.2.2 Experimental Results . . . 68
4.3 Summary . . . 71

5 From Racing to Online Hyper-Evolution 73
5.1 Online Racing and Population Cloning in Multirobot Systems . . . 74
5.1.1 Background . . . 74
5.1.2 Implementing Racing and Cloning in Multirobot Systems . . . 75
5.1.3 Experimental Methodology . . . 76
5.1.4 Experimental Results . . . 77
5.2 Introducing Online Hyper-Evolution . . . 82
5.2.1 Background . . . 82
5.2.2 Online Hyper-Evolution . . . 84
5.2.3 Experimental Results . . . 87
5.2.4 Improving the Hyper-Level . . . 92

5.3 Summary . . . 96

6 Online Behaviour Learning and Adaptation in Real Robots 97

6.1 Experimental Methodology . . . 98

6.1.1 Navigation and Obstacle Avoidance . . . 100

6.1.2 Homing . . . 100

6.1.3 Aggregation in a Non-determined Area . . . 101

6.1.4 Adaptation Experiments . . . 102

6.1.5 Performance Analysis and Treatments . . . 102

6.2 Experimental Results . . . 103

6.2.1 Transferring Simulation-evolved Controllers to Real Robots . . . 103

6.2.2 Evolution of Control in Real Robots . . . 106

6.2.3 Behaviour Adaptation and Fault Tolerance in Real Robots . . . 109

6.2.4 Ablation Studies . . . 111

6.3 Summary . . . 112

7 Conclusions and Future Work 113

7.1 Future Work . . . 114

7.1.1 Self-supervised Behaviour Learning . . . 114

7.1.2 Improving Online Hyper-Evolution . . . 114

7.1.3 Scaling the Evolutionary Process to More Complex Tasks . . . 115

7.1.4 Recovering from Faults and Damage via Evolution of Robot Body Plans . . . 115

A The Role of the Genomic Encoding in the Evolution of Complex Controllers 117

A.1 Evolving Complex Controllers with Indirect Encodings . . . 117

A.2 Hybridising Encodings . . . 119

Bibliography 121

List of Figures

1.1 Real robotic platform used in Chapter 6. Each Thymio II robot is extended with a Raspberry Pi 2 B single-board computer. . . . 7

2.1 Schematic illustration of combined body plan and control system specification . . . 14
2.2 Driving the search in evolutionary robotics experiments. . . . 16
2.3 A maze with obstacles that prevent a direct route from the robot to the goal. . . . 17
2.4 A chronogram of studies addressing online evolution . . . 26
2.5 Common non-linear neuron activation functions . . . 28
2.6 Example of a neural network with two input neurons, three hidden neurons, and two output neurons. . . . 29
2.7 Recurrent and feedforward neural network architectures . . . 29
2.8 The variable length genome problem . . . 31
2.9 A NEAT genotype to phenotype mapping example . . . 32
2.10 Matching up genomes for different network topologies in NEAT . . . 33
2.11 Complexification policy in NEAT . . . 34

3.1 Sammon's mapping of genotypes evolved . . . 48
3.2 Fault injections during the aggregation task: average operation time (age) of controllers . . . 53
3.3 Summary of the results concerning the injection of faults in the robots' sensors. . . . 54

4.1 The two types of environments, plain and complex, used to evolve controllers. . . . 61
4.2 Combined Sammon's mapping of intermediate genotypes. . . . 65
4.3 Distribution of the fitness score of the final controllers . . . 69
4.4 Distribution of evaluations . . . 70
4.5 Operation time of intermediate controllers in the concurrent foraging task. . . . 71

5.1 Mean fitness score of controllers throughout the simulation trials. . . . 77
5.2 Distribution of the mean group fitness of the final controllers. . . . 78
5.3 Distribution of the RSD of the final controllers. . . . 79
5.4 Mean fitness score of controllers throughout the simulation trials for the first set of complementary experiments . . . 80
5.5 Mean fitness score of controllers throughout the simulation trials for the second set of complementary experiments . . . 80

5.6 Mean operation time of controllers produced throughout the simulation trials for the second set of complementary experiments . . . 81
5.7 Illustration of a hyper-heuristic system. . . . 83
5.8 Illustration of the multi-level selective pressure in online hyper-evolution. . . . 87
5.9 Mean fitness score of controllers evolved by base-level algorithms and by **OHE-fitness+fit** throughout the simulation trials . . . 88
5.10 Mean number of algorithm instances used by **OHE-fitness+fit** throughout evolution in the **p1** setup and in the **p10** setup. . . . 89
5.11 Mean fitness score of controllers evolved by the different OHE techniques throughout the simulation trials . . . 90
5.12 Mean amount of time per robot that each algorithm was used by **OHE-fitness+fit** . . . 91
5.13 Sammon's mapping for the **p5** setup . . . 92
5.14 Analysis of successful runs in the algorithm selection experiments . . . 94
5.15 Analysis of successful runs in the algorithm construction experiments . . . 96

6.1 Experimental methodology and real robotic platform . . . 99
6.2 Real environment for the homing experiments. . . . 101
6.3 Real environment for the aggregation and adaptation experiments. . . . 101
6.4 Fitness plots for the simulation-based experiments . . . 105
6.5 Behaviour-fitness map for the navigation and obstacle avoidance tasks . . . 107
6.6 Successful controllers for the homing task . . . 108
6.7 Successful controllers for the aggregation task . . . 109
6.8 Successful controllers for the adaptation experiments . . . 110
6.9 Behaviour-fitness map for the adaptation experiments . . . 111
6.10 Successful controllers for the ablation experiments . . . 112

A.1 HyperNEAT connectivity patterns production . . . 118
A.2 R-HybrID connectivity patterns production . . . 120

List of Tables

2.1 Summary of online evolution approaches introduced starting from the point embodied evolution was developed . . . 27

3.1 Summary of the main differences between odNEAT, rtNEAT, and IM-$(\mu + 1)$ . . . 39
3.2 Summary of the experimental details . . . 43
3.3 Aggregation controller details . . . 44
3.4 Navigation and obstacle avoidance controller details . . . 45
3.5 Phototaxis controller details . . . 46
3.6 Comparison of the number of evaluations and of the fitness scores of solutions to the task between odNEAT and rtNEAT . . . 47
3.7 Summary of the neural complexity added through evolution from the initial topology by odNEAT and rtNEAT . . . 49
3.8 Generalisation performance of controllers evolved by odNEAT and rtNEAT in the three tasks . . . 50
3.9 Comparison of the number of evaluations and of the fitness scores of solutions to the task (out of 100) between odNEAT and IM-$(\mu + 1)$ . . . 50
3.10 Summary of the neural complexity added through evolution from the initial topology by odNEAT and IM-$(\mu + 1)$ . . . 51
3.11 Generalisation performance of controllers evolved by odNEAT and IM-$(\mu + 1)$ in the three tasks 52
3.12 odNEAT ablations summary (simulation time, number of evaluations, and success rate for each experimental configuration) . . . 55

4.1 Number of evaluations for the three types of controllers considered . . . 63
4.2 Neural complexity of solutions evolved . . . 63
4.3 Generalisation performance of each controller model . . . 64
4.4 Summary of inter-behaviour distances and intra-behaviour distances between final controllers to the task. . . . 66
4.5 Controller details and sensor ranges . . . 67

5.1 Success rates for the foraging tasks in the algorithm selection experiments . . . 94
5.2 Success rates for the foraging tasks in the algorithm construction experiments . . . 95

6.1 Controller performance in simulation and in reality . . . 104

Chapter 1

Introduction

The idea of autonomous machines has stirred human imagination since ancient history and mythology (Rossi et al., 2009). For example, tales of the Greek god Hephaestus mentioned three-legged tables that walked by themselves (Bedini, 1964; Rossi et al., 2009), while the treatises of inventors such as Heron of Alexandria (10 AD to 70 AD) described designs of mechanical ducks and birds. Later, in the Renaissance, the famed artist and inventor Leonardo da Vinci designed a mechanical knight that could stand, sit up, and independently move its arms and legs, and a mechanical lion that could walk, move the head, shake the tail, and open its jaws (Rosheim, 2006). The increasing interest in such machines, accompanied by the rise of electronics in the 20th century, developed into what is now called a robot: "a machine capable of carrying out a complex series of actions automatically, especially one programmable by a computer" (Oxford English Dictionary).

Robots have changed the world, and can currently be found in multiple sectors across society (Gates, 2007; Bekey and Yuh, 2008). Examples include industrial robots able to operate continuously in factories, domestic robots that can perform daily chores, assistive robots for people with disabilities (e.g. robotic walking assistance devices), and robotic surgical systems designed to assist surgeons with complex medical operations (Bekey and Yuh, 2008). Arguably, some of the main achievements in robotics have been carried out by robots with high *decisional autonomy* (Bellingham and Rajan, 2007). According to the Robotics 2020 Multi-Annual Roadmap (SPARC, 2015), decisional autonomy refers to the ability of a robot to operate autonomously, that is, to assess its own situation in the context of the mission, and to decide how to behave in order to accomplish the mission or task objectives. All robot systems have some degree of autonomy, which ranges from the simple motion of an assembly line stopped by a sensor reading, to the ability to be self-sufficient in a complex environment. At the highest levels of autonomy, a robot can dynamically alter its tasking, and carry out different actions in response to dynamic real-time events in the environment. Current highly-autonomous robots can, for example, roam the surface of other planets to aid in scientific discoveries (Squyres et al., 2004; Bellingham and Rajan, 2007).

One notable highly-autonomous robot was NASA's Deep Space 1 spacecraft. Sixty-five million miles from Earth (Bellingham and Rajan, 2007), Deep Space 1 was able to autonomously diagnose failures and recover from simulated faults remotely injected by human engineers. The faults were injected while the spacecraft was flying towards 9969 Braille, a small Mars-crossing asteroid, and enabled assessing the ability of Deep Space 1 to correct its own trajectory. Furthermore, Deep Space 1 was able to autonomously navigate by using pattern recognition to compare images of stars with a known star catalogue, and thus triangulate its position during the interplanetary

cruise. While the autonomous mission capabilities of Deep Space 1 have impressed, the robot's control system is based on a highly-specialised artificial intelligence system called *remote agent* (Muscettola et al., 1998). Remote agent is an implementation of the sense-plan-act model driven by model-based programming, and constraint-based planning and scheduling. The behaviour of Deep Space 1 is therefore limited to manually-defined rules and specifications that remain fixed during task execution, regardless of what the robot experiences during its mission. Like most robots, Deep Space 1 is therefore a *narrow artificial intelligence* system unable to adapt to unanticipated conditions. For example, a fault in the star tracking system required the robot to stop its mission for approximately eight months, while engineers developed new ways for the robot to navigate.[1]

In order to enable the development of intelligent, reliable, mobile robots that can effectively operate in a wide variety of environments, four areas of research are considered crucial (Bellingham and Rajan, 2007), namely: (i) software engineering capabilities for verification and validation, (ii) novel methods for representing and reasoning about uncertainty, (iii) better sensing capabilities, and finally (iv) online learning during task execution, which is the topic of this research. The key principle behind online learning is that robots can accumulate experience over time in order to make increasingly more informed and effective decisions, and thus become better equipped to deal with unstructured environments and unforeseen conditions. In this way, robots can achieve high levels of *adaptability*, which is closely related with decisional autonomy. In the Robotics 2020 Multi-Annual Roadmap (SPARC, 2015), adaptability is described as the ability of a system to self-adapt to different work scenarios, environments, and conditions, both over short and long time-scales. Examples of adaptability include modifying the parameters of a controller to account for changes in the sensorimotor apparatus, such as a failed sensing element, increased friction or reduced power in one or more actuators; adapting to a new environment by changing the behavioural strategy used to carry out a task. In this respect, adaptability implies that the robot carries out an optimisation process with respect to some performance or evaluation criteria.

Online learning involves learning new behaviours and/or adapting existing behaviours. Online learning methodologies can be divided into (Brooks and Matarić, 1993):

- Weak learning techniques, where the robot has numerous built-in behavioural capabilities or significant *a priori* knowledge. The structure and/or type of the built-in capabilities or knowledge restricts what the robot can learn.

- Strong learning techniques, which assume no or little built-in behavioural capabilities or knowledge. In this way, the capabilities and knowledge required to carry out a task must be acquired online. While little built-in structure reduces the learning bias and eases the experimenter's task of preparing a set of robots to carry out a task, it can also increase the amount of time required for robots to learn how to solve a task. This trade-off between the amount of built-in behavioural capabilities and learned behaviours is a key issue in online learning (Brooks, 1992).

Online learning is well-aligned with a long-standing goal in artificial intelligence and robotics: synthesising artificial agents that can effectively learn and adapt throughout their lifetime (Russell and Norvig, 1995). The development of general-purpose robots capable of learning a variety of different skills, and of applying them to

[1] http://www.jpl.nasa.gov/missions/deep-space-1-ds1/

solve numerous tasks, would enable robots to be actively involved in our daily lives, and to more effectively carry out tasks in environments that are either remote or hazardous for man, such as space, underground, or deep sea (Bellingham and Rajan, 2007).

Despite the progress, general-purpose robots remain elusive for a number of reasons. Online learning is intended to increase the effectiveness and appropriateness of robots' actions. This ambitious goal can hardly be achieved with weak learning approaches because of the restrictions they impose on the learning process (Brooks and Matarić, 1993). Strong learning techniques, on the other hand, may require a significant amount of time to learn effective behaviours because they tend to operate in large spaces of possible behaviours, and may thus require a large number of trials to find suitable solutions (Brooks and Matarić, 1993; Silva et al., 2016d). Furthermore, the *type* of learning technique also plays a crucial role in robotics. Several techniques are dependent on the type of robot (e.g. if the robot is wheeled or legged) (Yosinski et al., 2011; Cully et al., 2015), while others cannot handle the often noisy evaluation conditions and sparse feedback that occur during online learning processes (Montanier, 2013).

In this thesis, we study new ways to enable efficient online learning in autonomous robots. We employ a control synthesis methodology called *evolutionary robotics* (Floreano and Mondada, 1994; Husbands et al., 1997). Evolutionary robotics emerged in the 1990s as a promising alternative to classic artificial intelligence techniques and traditional control systems. In general terms, evolutionary robotics approaches maintain a population of genomes, each encoding the control system for one or more robots. For example, if the control system is an artificial neural network (ANN), the connection weights can be represented at the genome level as a real-valued vector, while finite state machine-based controllers can be described by automaton-based representations (König et al., 2009). Although less frequent, evolutionary robotics techniques can also be applied to generate robot body plans, and to coevolve control systems and body plans simultaneously (Lipson and Pollack, 2000). Optimisation of genomes is based on Darwin's theory of evolution, namely blind variations and survival of the fittest, as embodied in the neo-Darwinian synthesis (Gould, 2002).

Evolutionary robotics is a highly general approach, as it enables the synthesis of control given only a specification of the task, and is not tied to specific evolutionary algorithms, control systems, or types of robots (Bongard et al., 2006; Cully et al., 2015). Furthermore, evolutionary robotics differs from more traditional approaches because it puts a strong emphasis on embodiment and situatedness, and on the close interaction of brain, body, and environment, which is considered crucial for the emergence of intelligent, adaptive behaviour and cognitive processes (Chiel and Beer, 1997; Clark, 1998; Nolfi and Floreano, 2002). Evolutionary robotics therefore has the potential to enable the automatic design of control systems without the need for manual and detailed specification of the desired behaviour (Floreano and Keller, 2010; Nolfi, 1998), and to exploit the way in which the world is perceived through the robots' sensors (Floreano and Mondada, 1994). In the early years of evolutionary robotics, contributions such as that of Harvey et al. (1993); Floreano and Mondada (1994) laid the foundation for a number of important studies that followed. These studies advocated the use of artificial neural networks as the underlying control system of autonomous robots (Floreano and Mondada, 1994; Husbands et al., 1997). The ANN paradigm was adopted due to a number of features such as the ANNs' robustness in dynamic, real-world environments (Floreano and Mondada, 1994), and the argued tight coupling between perception and action (Husbands et al., 1997).

In online learning through evolutionary robotics techniques, henceforth called *online evolution*, an evolutionary algorithm is executed onboard each robot in order to create and continuously optimise its behavioural control logic. Each instance of the evolutionary algorithm executes without any external supervision or human intervention. Online evolution can thus automatically generate the artificial intelligence that controls each robot. That is, online evolution creates the potential for long-term behaviour adaptation and learning: robots can continuously self-adjust and learn new behaviours in response to, for example, changes in the task requirements or environmental conditions, and to faults in the sensors or actuators (Silva et al., 2015e, 2017a).

The first studies on online evolution were conducted by Floreano and Mondada (1994, 1996), who experimented with evolution of controllers directly in real robotic hardware, including navigation and homing behaviours for a Khepera robot. Given the low computational power of the Khepera robot, the actual evolutionary computation was performed on a workstation. Whereas the pioneering contributions of Floreano and Mondada (1994, 1996) used a single robot, groups of robots can be beneficial to online evolution. The fact that multiple robots operate simultaneously means that they can *learn in parallel*, and exchange candidate solutions to the task when they meet, thus facilitating collective problem solving (Watson et al., 2002). Similarly to the learning process, behavioural control in a multirobot system is decentralised: each individual robot operates based on its local observations of the environment and on the interaction with robots nearby. Decentralised control implies that there is no single point of failure in the system, and enables individual robots to carry out tasks involving monitoring, searching, or collecting data in an independent manner.

1.1 Problem Statement

Despite the potential for automatic robot learning, online evolution is not frequently employed for a number of reasons. First, online evolution typically requires several hours or days to synthesise solutions to a task (Silva et al., 2016d). As a result, the approach has not yet been practically exploited in real-robot systems. Second, one common assumption in the field is that online evolution enables continuous learning and adaptation to previously unforeseen circumstances. However, only a small number of ad-hoc experiments have been carried out in simulation (Bredeche et al., 2012; Silva et al., 2015e). In real robotic hardware, demonstrations of the adaptive capabilities of online evolution have been limited to manually changing the position of a light source in a phototaxis task (Bredeche et al., 2012). That is, the potential for online evolution to enable dynamic adaptation and learning has been largely left unstudied.

The main goal of this thesis is to address some of the fundamental issues associated with online evolution to bring it closer to widespread adoption. In particular, we focus on the following research questions: (i) how to minimise the intervention required from the human designer in order to facilitate open-endedness in online evolution approaches, (ii) how to accelerate and increase the performance of online evolution at the individual robot level and at group level, and (iii) given that simulations are extensively used in the field, how can we enable efficient behaviour learning and adaptation in real robots. Throughout our research, we have proposed and studied multiple approaches to: (i) cut short the amount of time required for effective behavioural control synthesis, and (ii) increase the performance and robustness of online evolution algorithms. Importantly, we have focused on developing methods that could additionally minimise the amount of intervention from the human designer. For example, previous approaches to online evolution of neural network-based controllers

typically required the experimenter to decide on a suitable controller structure beforehand, which involves a significant amount of experimentation and knowledge. Our first research contribution was the development and assessment of an algorithm called odNEAT (Silva et al., 2015e) in which both the weights and the topology of neural networks are under evolutionary control, and an appropriate degree of complexity to solve the current task is the result of a continuous evolutionary process.

odNEAT is at the heart of our research. Throughout this research, the algorithm was extensively assessed in simulation-based experiments, extended with novel techniques such as the capability to cut short the evaluation of poor controllers, and then validated in real-robotic hardware. On real robots, odNEAT evolved solutions to three classic evolutionary robotics tasks in a timely manner (one hour or less), and enabled fault tolerance in groups of robots. Our main research contributions are summarised in the following section, and resulted in seven journal articles, seven conference papers, and one workshop paper.

1.2 Thesis Structure and Contribution of Research

In this section, we provide an overview of the thesis structure and the scientific publications produced over the past four years leading to this thesis. In Chapter 2, we cover the background on the key topics related to our contributions. The chapter is divided in four major sections, namely: (i) introduction to the field of evolutionary robotics, including early pioneering contributions and open issues in evolutionary robot engineering, (ii) review and discussion of the state of the art in online evolution of behavioural control for robots, (iii) overview of fundamental concepts regarding artificial neural networks theory, and (iv) analysis of developments on evolutionary optimisation of artificial neural networks. The first two sections of the chapter are based on the following articles:

- F. Silva, L. Correia, and A. L. Christensen (2016). **Evolutionary Robotics**, *Scholarpedia*, 11(7):33333.
- F. Silva, M. Duarte, L. Correia, S. M. Oliveira, and A. L. Christensen (2016). **Open Issues in Evolutionary Robotics**, *Evolutionary Computation*, 24(2), 205–236.

In Chapter 3, we provide a comprehensive presentation and analysis of Online Decentralised NeuroEvolution of Augmenting Topologies (odNEAT), initially introduced in Silva et al. (2012). odNEAT is an algorithm for online evolution of artificial neural network-based controllers in multirobot systems. odNEAT is distributed across multiple robots that evolve in parallel and exchange candidate solutions to the task when they meet. odNEAT differs from more traditional approaches to online evolution because both the weighting parameters and the topological structure of neural networks are under evolutionary control. The work presented in the chapter resulted in the following publication:

- F. Silva, P. Urbano, L. Correia, and A. L. Christensen (2015). **odNEAT: An Algorithm for Decentralised Online Evolution of Robotic Controllers**, *Evolutionary Computation*, 23(3), 421-449.

In Chapter 4, we report simulation-based experiments designed to investigate the dynamics of online evolution of controllers at two different scales. At the microscopic scale, we study the dynamics of distinct neuronal models. At the macroscopic scale, we investigate the scalability properties of online evolution with respect to group size. The work presented in the chapter resulted in the following publications:

- F. Silva, L. Correia, and A. L. Christensen (2013). **Dynamics of Neuronal Models in Online Neuroevolution of Robotic Controllers**. In: *16th Portuguese Conference on Artificial Intelligence*, pages 90–101. Springer, Berlin, Germany.
  Nominated for the Best Student Paper Award. Nominated for the Best Paper Award.

- F. Silva, L. Correia, and A. L. Christensen (2015). **A Case Study on the Scalability of Online Evolution of Robotic Controllers**. In: *17th Portuguese Conference on Artificial Intelligence*, pages 189–200. Springer International Publishing, Switzerland.

The approaches reported in Chapter 3 and in the first part of Chapter 4 have originated two novel approaches that are not presented in this thesis. In such approaches, the online evolutionary process has been respectively combined with: (i) a biologically-inspired form of synaptic plasticity called neuromodulation (Katz, 1999; Bailey et al., 2000), and with (ii) behavioural building blocks called *macro-neurons* that enable the specification of evolved or preprogrammed behaviours in the neural architecture. These studies resulted in the following publications:

- F .Silva, P. Urbano, and A. L. Christensen (2014). **Online Evolution of Adaptive Robot Behaviour**, *International Journal of Natural Computing Research*, 4(2):59–77.

- F. Silva, L. Correia, and A. L. Christensen (2014). **Speeding up Online Evolution of Robotic Controllers with Macro-neurons**. In: *17th European Conference on the Applications of Evolutionary Computation*, pages 765–776. Springer, Berlin, Germany.

- F. Silva, M. Duarte, S. M. Oliveira, L. Correia, and A. L. Christensen (2014). **The Case for Engineering the Evolution of Robot Controllers**. In: *14th International Conference on the Synthesis and Simulation of Living Systems*, pages 703–710. MIT Press, Cambridge, MA.

In Chapter 5, we present two simulation-based case studies. In the first case study, we introduce two novel approaches to accelerate online evolution of robotic controllers in multirobot systems, namely a racing approach for multirobot systems that allows individual robots to cut short the evaluation of poor controllers, and a population cloning approach that enables each individual robot to clone and transmit a varying number of high-performing controllers stored in its internal population to other robots nearby. The underlying motivation is to effectively leverage the genetic information accumulated by multiple robots that evolve together. In the second case study, we introduce *online hyper-evolution* (OHE), in which the key principle is to use the different sources of feedback information traditionally associated with controller evaluation, namely task performance (fitness) or behaviour (novelty or diversity), in order to select and/or construct suitable evolutionary algorithms online. Contrarily to current approaches to online evolution, which rely on a single, predefined algorithm to find a high-performing controller, OHE searches across a space of candidate algorithms and configurations in order to find effective algorithms over time. The two case studies resulted in the following publications:

- F. Silva, L. Correia, and A. L. Christensen (2016). **Leveraging Online Racing and Population Cloning in Evolutionary Multirobot Systems**. In: *19th European Conference on the Applications of Evolutionary Computation*, pages 165–180. Springer International Publishing, Switzerland.
  Best Paper Award (Evolutionary Robotics track).

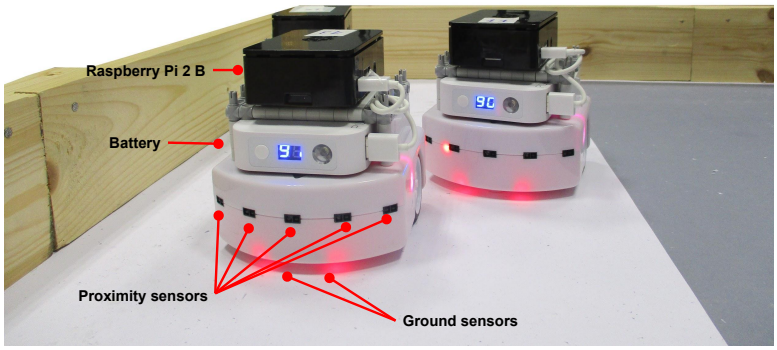


Figure 1.1: Real robotic platform. Each Thymio II robot is extended with a Raspberry Pi 2 B single-board computer. The Thymio II robot has seven infrared proximity sensors (five in the front, two in the back) and two infrared ground sensors, among others.

- F. Silva, L. Correia, and A. L. Christensen (2016). **Online Hyper-Evolution of Robotic Controllers in Multirobot Systems**. In: *10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 11–20. IEEE Computer Society, Los Alamitos, CA.
  Best Student Paper Award.

- F. Silva, L. Correia, and A. L. Christensen (2017). **Hyper-Learning Algorithms for Online Evolution of Robot Controllers**. *ACM Transactions on Autonomous and Adaptive Systems*. Invited contribution, to appear.

In Chapter 6, we report successful evolution on neural network-based controllers in real robotic hardware to solve two single-robot tasks and one collective robotics task. We used Thymio II robots (`https://www.thymio.org/en:thymio`). Each robot was extended with a Raspberry Pi 2 B single-board computer, as shown in Figure 1.1. For the three tasks, controllers were evolved either from random solutions or from solutions pre-evolved in simulation. In all cases, capable solutions were found in a timely manner (one hour or less). We furthermore demonstrate for the first time the adaptive capabilities of online evolution in real robotic hardware, including a robot collective able to overcome faults injected in the motors of multiple robots simultaneously. We conclude by showing that one key enabler of the robots' ability to effectively learn and adapt is the underlying evolutionary algorithm, odNEAT. The work has been published in the journal Royal Society Open Science:

- F. Silva, L. Correia, and A. L. Christensen (2017). **Evolutionary Online Behaviour Learning and Adaptation in Real Robots**. *Royal Society Open Science*, 4:160938.

In Chapter 7, we provide an overview of our contributions and discuss directions for future research. Summaries of our contributions at different stages of our research can be found in the following contributions:

- F. Silva, L. Correia, and A. L. Christensen (2014). **Towards Online Evolution of Control for Real Robots with odNEAT**, *3rd International Workshop on the Evolution of Physical Systems, held as part of the 14th International Conference on the Synthesis & Simulation of Living Systems.*

- F. Silva, A. L. Christensen, and L. Correia (2015). **Engineering Online Evolution of Robot Behaviour**. In: *International Conference on Autonomous Agents and Multiagent Systems*, pages 2017–2018. IFAAMAS, Richland, SC.

- F. Silva, L. Correia, and A. L. Christensen (2017). **Evolutionary Online Learning in Multirobot Systems**, *AI Matters*, 3(1):23–24.

1.3 Other Scientific Contributions

During our research, we have conducted a number of studies not directly related to the main topic of this thesis. These studies have resulted in one book chapter, two journal articles, eight conference publications, and one peer-reviewed video. The contributions are related with:

- One of the open issues in evolutionary robotics, namely the design of genomic encodings that facilitate the evolution of complex behaviours and large-scale controllers, see Section 2.1.4 and Appendix A for details.
  - F. Silva, L. Correia, and A. L. Christensen (2015). **R-HybrID: Evolution of Agent Controllers with a Hybridisation of Indirect and Direct Encodings**. In: *International Conference on Autonomous Agents and Multiagent Systems*, pages 735–744. IFAAMAS, Richland, SC.
- The development of suitable simulation platforms for research and education in evolutionary robotics, which is seldom emphasised in scientific publications (Silva et al., 2016d).
  - M. Duarte, F. Silva, T. Rodrigues, S. M. Oliveira, and A. L. Christensen (2014). **JBotEvolver: A Versatile Simulation Platform for Evolutionary Robotics**. In: *14th International Conference on the Synthesis and Simulation of Living Systems*, pages 210–211. MIT Press, Cambridge, MA.
- Collective behaviour synchronisation in groups of stationary robots in the context of the European Union – Information and Communication Technologies project 'ASSISIbf', no 601074, on mixed animal-robot societies, see Silva et al. (2015c) for details.
  - F. Silva, L. Correia, and A. L. Christensen (2015). **Modelling Synchronisation in Multirobot Systems with Cellular Automata: Analysis of Update Methods and Topology Perturbations**. In: *Robots and Lattice Automata*, eds. G. Sirakoulis and A. Adamatzky. Emergence, Complexity and Computation 13, 267–293.
  - R. Mills, P. Zahadat, F. Silva, D. Miklic, P. Mariano, T. Schmickl, and L. Correia (2015). **Coordination of Collective Behaviours in Spatially Separated Agents**. In: *13th European Conference on Artificial Life*, pages 579–586. MIT Press, Cambridge, MA.
- Evolution of control for swarms of aquatic surface robots in the scope of the CORATAM and HANCAD projects (Control of Aquatic Drones for Maritime Tasks, FCT project EXPL/EEIAUT/0329/2013; Heterogeneous Ad-hoc Network for the Coordination of Aquatic Drones, internal project from the Instituto de Telecomunicações), see Christensen et al. (2015) for details.
  - M. Duarte, V. Costa, J. Gomes, T. Rodrigues, F. Silva, S. M. Oliveira, and A. L. Christensen (2016). **Evolution of Collective Behaviors for a Real Swarm of Aquatic Surface Robots**. *PLoS ONE* 11(3),e0151834.

  - M. Duarte, V. Costa, J. Gomes, T. Rodrigues, F. Silva, S. M. Oliveira, and A. L. Christensen (2016). **Unleashing the Potential of Evolutionary Swarm Robotics in the Real World**. In: *Genetic and Evolutionary Computation Conference*, pages 159–160. ACM Press, New York, NY.
  - M. Duarte, J. Gomes, V. Costa, T. Rodrigues, F. Silva, V. Lobo, M. Marques, S. M. Oliveira, and A. L. Christensen (2016). **Application of Swarm Robotic Systems to Marine Environmental Monitoring**. In: *IEEE/MTS OCEANS*, pages 1–8. IEEE Press, Piscataway, NJ.
  - A. L. Christensen, S. M. Oliveira, O. Postolache, M. J. de Oliveira, S. Sargento, P. Santana, L. Nunes, F. Velez, P. Sebastião, V. Costa, M. Duarte, J. Gomes, T. Rodrigues, and F. Silva (2015). **Design of Communication and Control for Swarms of Aquatic Surface Drones**. In: *7th International Conference on Agents and Artificial Intelligence*, pages 548–555. SCITEPRESS, Lisbon, Portugal.
  - F. J. Velez, A. Nadziejko, A. L. Christensen, S. M. Oliveira, T. Rodrigues, V. Costa, M. Duarte, F. Silva, and J. Gomes (2015). **Wireless Sensor and Networking Technologies for Swarms of Aquatic Surface Drones**. In: *IEEE 82nd Vehicular Technology Conference*, pages 1–2. IEEE Press, Piscataway, NJ.
  - F. J. Velez, A. Nadziejko, A. L. Christensen, S. M. Oliveira, T. Rodrigues, V. Costa, M. Duarte, F. Silva, and J. Gomes (2015). **Experimental Characterization of WSNs Applied to Swarms of Aquatic Surface Drones**. In: *10th Conference on Telecommunications*. Unpublished.
  - A. L. Christensen, M. Duarte, V. Costa, T. Rodrigues, J. Gomes, F. Silva, and S. M. Oliveira (2016). **A Sea of Robots**. In: *AAAI Video Competition 2016, held as part of the 30th AAAI Conference on Artificial Intelligence*, Phoenix, Arizona, US.
    Best Robot Video award. URL: `https://goo.gl/Qxgi12`.

- Modelling discrete dynamical systems under asynchrony and noise.

  - F. Silva and L. Correia (2013). **An Experimental Study of Noise and Asynchrony in Elementary Cellular Automata with Sampling Compensation**. *Natural Computing*, 12(4), 573–588.

1.4 Software Tools

During our research, we conducted experiments in simulation and on real robotic hardware. We have therefore developed and extended numerous software tools. Below, we provide an overview of the main software tools used, which are available under the GNU GPL licence, and can be found online at `https://github.com/fgsilva`.

1.4.1 Simulation Platform

The significance of suitable simulation platforms in seldom emphasised in scientific publications on robotics (Silva et al., 2016d). To conduct our simulation-based experiments, we extended JBotEvolver (Duarte et al., 2014) to support online evolution and online learning, and actively participated in the continuous development of the platform. JBotEvolver is an open-source, cross-platform framework designed for research and education in evolutionary robotics. JBotEvolver is developed in Java, and has been used in over 40 evolutionary robotics

studies of our research group (see `https://github.com/BioMachinesLab/jbotevolver`), and in a number of undergraduate and graduate courses at the University Institute of Lisbon (ISCTE-IUL).

JBotEvolver's main features are its ease of installation and use, and its versatility in terms of customisation and extension. A fundamental design philosophy behind JBotEvolver is to provide a basis for evolutionary robotics experiments without the need for detailed framework-specific knowledge: JBotEvolver enables the configuration of experiments programmatically or via a plaintext file that specifies which features will be included in the simulation. The corresponding classes are then seamlessly loaded in execution time via Java's Reflection API. In this way, JBotEvolver can also make use of external, user-defined classes that extend the base implementation. Additionally, JBotEvolver is self-contained, but can also be used as an external library in other applications, and enables experiments to be defined as tasks that can be executed sequentially or in parallel on a workstation, or on distributed computing platforms (Duarte et al., 2014).

1.4.2 Common Interface, Raspberry Controller, and Robot Control Console

In order to effectively conduct our real-robot experiments, we extended the control software originally developed in the context of the CORATAM and HANCAD projects (Christensen et al., 2015) for aquatic surface robots.

- **Common Interface** – We developed a middle-layer that is shared by both JBotEvolver and the onboard software of the real robots. This approach enables the same code-base, control systems, and evolutionary algorithms to be executed in both simulated robots and real robots. As in the original Common Interface, our implementation defines communication, sensing, and actuating components only once, therefore minimising implementation differences between simulation-based experiments and real-robot experiments.
- **Raspberry Controller** – This software component is used to access the hardware layers of the real robots. The Common Interface employs the Raspberry Controller in order to: (i) access data from the robot's hardware (e.g. retrieving sensors readings), (ii) send data packets via the wireless network module to other robots or to a control centre, and (iii) actuate the robot's motors. We extended the original Raspberry Controller to support Thymio II robots and new sensors mounted on the Raspberry Pi (e.g. camera), and we implemented the low-level communication between the Thymio II robots and the Raspberry PI, and between the Raspberry PI and the Raspberry Controller.
- **Robot Control Console** – The control software is additionally composed of a GUI that allows a user to start experiments, deploy controllers to groups of robots, access data from the robots, such as speed, orientation and other diagnostics information, and update and restart a robot's onboard control software. The robot control console was only subject to minor modifications.

1.5 Summary

In this chapter, we have motivated online learning and online evolution in autonomous robots. The development of reliable and intelligent machines requires that robots can learn new behaviours and adapt their behaviour in response to unforeseen conditions. We stated the main topic of this research, namely online evolution in groups of autonomous robots. We listed the original research and the publications in which this thesis is based. We then briefly presented original research related to other topics in which we made scientific contributions: design

of genomic encodings that facilitate the evolution of complex behaviours and large-scale controllers, suitable simulation platforms for research and education in evolutionary robotics, collective behaviour synchronisation in groups of stationary robots, evolution of control for swarms of aquatic surface robots, and modelling discrete dynamical systems. To conclude, we described the main software tools used in our research.

Chapter 2

Background

In this chapter, we cover the background on the key topics related to our contributions. We first present the main concepts of evolutionary robotics. We then discuss the state of the art in online evolution, introduce fundamental concepts regarding artificial neural networks theory, and review recent developments on fixed-topology and constructive neuroevolution algorithms.

2.1 Introduction to Evolutionary Robotics

Evolutionary algorithms (EAs) have been the subject of significant progress since the introduction of the concept of evolutionary search by Turing (1950). Early pioneering work includes Barricelli's seminal contributions and computational experiments since 1953 on symbiogenesis (see Barricelli, 1962; Fogel, 2006), and the studies by Fraser (1957) on the effects of selection in epistatic systems. Afterwards, the introduction of genetic algorithms by Holland (1962) as a computational abstraction of Darwin's theory of evolution promised to transfer the adaptation capabilities of natural organisms to different types of artificial agents, including autonomous robots. The envisioned possibilities inspired a new field of research, now called *evolutionary robotics* (ER) (Nolfi and Floreano, 2000). Evolutionary robotics employs evolutionary computation techniques to generate robots that adapt to their environment through a process analogous to natural evolution. The generation and optimisation of robots are based on evolutionary principles of blind variations and survival of the fittest, as embodied in the neo-Darwinian synthesis (Gould, 2002).

Evolutionary robotics is organised along two axes of research: one concerned with cognitive science and biology (Harvey et al., 2005; Floreano and Keller, 2010); the other focused on using evolutionary robotics techniques for engineering purposes (Silva et al., 2016d), with the long-term goal of obtaining a process capable of automatically designing and maintaining an efficient robotic system. In this respect, the use of evolutionary principles has been largely argued as necessary to replace preprogrammed approaches (Nolfi and Floreano, 2000; Harvey et al., 1997; Lipson and Pollack, 2000; Quinn et al., 2003).

2.1.1 Basic Framework

Similarly to more traditional evolutionary computation approaches, evolutionary robotics techniques operate with a population of candidate solutions or *genomes*. Each genome in the population encodes a number of parameters of one or more robots' body plan or control system, the *phenotype* (see Figure 2.1). If the genome

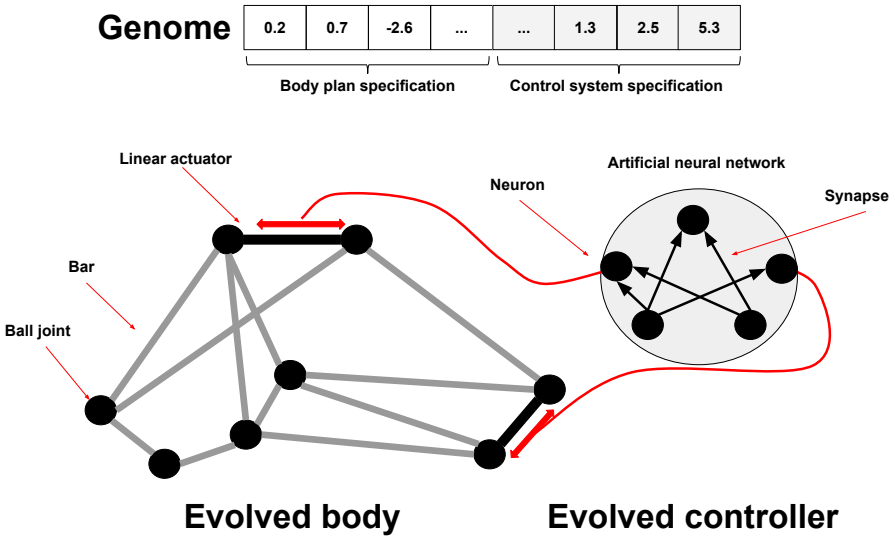


Figure 2.1: Schematic illustration of combined body plan and control system specification, as described in Lipson and Pollack (2000). Evolution can connect parts to each other to form arbitrary trusses (body plan), and can connect neurons to each other via synapses to form arbitrary neural networks (control systems). Neurons also connect to bars: in the same way that a real neuron governs the contraction of muscle tissue, the artificial neuron controls the length of the bar via a linear actuator. No sensors are used. As a result, these robots can generate patterns of actions, but cannot directly sense their environment.

describes a body plan, genome parameters can determine whether body parts should be added to the body plan, or configure parts of the body plan, such as the angle and range for specific joints (Bongard, 2011). If the genome encodes an artificial neural network-based controller, the synaptic weights can be represented as a real-valued vector at the genome level. If the genome encodes a finite state machine-based controller, an automaton-based representation can be used (König et al., 2009). The mapping from genotype to phenotype can capture different properties of the developmental process of natural organisms, and the phenotype can abstract various degrees of biological realism (Stanley and Miikkulainen, 2003). Evolutionary robotics thus draws inspiration from biological principles at multiple levels.

Evolutionary robotics techniques can be applied either offline, in simulation, or online, that is, on the physical robots while they operate in the task environment. In offline evolution, controllers are evolved in simulation for a certain number of generations or until a certain performance criterion is met, and then deployed on real robots. In online evolution, on the other hand, an evolutionary algorithm is executed on the robots themselves as they perform their tasks. Evolution is most commonly applied offline for a number of practical and strategic reasons. Firstly, offline evolution is typically less time consuming than online evolution, although suitable simulation environments have to be developed beforehand. Secondly, offline evolution allows the researcher to concentrate on developing the body plan or control method without having to address issues inherently associated with physical robots, such as wear and tear, potential damage, calibration drift, and so on.

2.1.2 Guiding the Search in Evolutionary Robotics

The experimenter applying evolutionary robotics techniques often relies on a self-organisation process in which evaluation and optimisation of controllers are holistic, thereby eliminating the need for manual and detailed

specification of the desired body plan or control system (Doncieux et al., 2011; Silva et al., 2016d). Below, we describe how to drive the evolutionary process in evolutionary robotics experiments.

Fitness-based Evolution

Similarly to other evolutionary methods, a traditional evolutionary robotics process requires only sparse feedback, which is given by a measure of overall performance, that is, a fitness score. The *fitness function* is therefore at the heart of multiple evolutionary robotics processes and rewards improvement towards a task-dependent objective[1], a metaphor for the pressure to adapt in nature.

In the literature, there are multiple categorisations of fitness functions (Nelson et al., 2009). According to the three-dimensional *fitness space framework* proposed by Floreano and Urzelai (2000), a fitness function can be classified along the following three axes:

- **Functional or behavioural** depending on whether the fitness function measures the components involved in the generation of the behaviour, such as the rotational speed of a robot's wheel in a navigation task, or the effects of the behaviour, such as the distance covered by a robot in a given amount of time.
- **External or internal** depending on whether the fitness computation is based on the measurement of variables that are only available to an external observer with access to precise information, such as the number of clusters formed by the robots in an aggregation task, or on information directly available to the robot through onboard sensor readings.
- **Explicit or implicit** depending on the quantity and nature of constraints explicitly imposed in the working principles of solutions. The higher the number of components, the more explicit the fitness function and the more constrained the behaviour is.

As discussed in Nolfi and Floreano (2000), a behavioural-implicit-internal approach may be preferable to design adaptive robots, because it imposes few biases on the search process, which in turn can lead to more efficient and effective behaviour adaptation.

Diversity-based Evolution

A class of methods[2] has recently emerged in evolutionary robotics in which evolution is driven by novelty or behavioural diversity instead of a typical fitness function, see Figure 2.2. The main idea is that candidate solutions can be scored based on a characterisation of their behaviour during evaluation, and not only based on a traditional fitness function. In a maze navigation task (see Figure 2.3), the behaviour of a controller could, for example, be characterised by: (i) the final position of the corresponding robot in the environment, which amounts to a characterisation of length 2, or (ii) n samples of the robot's (x, y) position taken at regular time intervals during an evaluation, in which case the resulting behavioural characterisation is the vector $(x_1, y_1, x_2, y_2, ..., x_n, y_n)$ (Lehman and Stanley, 2011a).

[1] Although less common, recent studies have conducted relevant work in the definition of task-independent fitness functions based on information theoretic measures such as Shannon's entropy or mutual information (Klyubin et al., 2005a,b; Prokopenko et al., 2006; Capdepuy et al., 2007; Sperati et al., 2008). Overall, these contributions indicate that, even though evolutionary robotics is historically considered an empirical endeavour, a theoretical foundation for the field could be developed (Bongard, 2013).

[2] There is no consensus concerning whether or not the behaviour should be considered part of the phenotype (Mouret and Doncieux, 2012). We distinguish between the concepts of *behaviour* and *phenotype* (often used as a synonym of *controller*), to make our description of behavioural diversity-based methods as clear as possible.

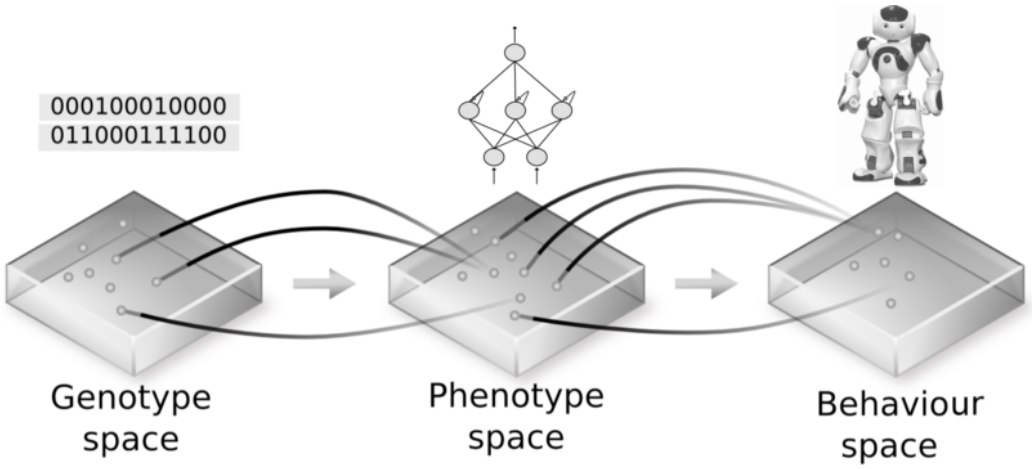


Figure 2.2: In traditional evolutionary algorithms, genomes of candidate solutions are translated into phenotypes whose fitness is assessed. In evolutionary robotics, the behaviour of the robot can also be characterised during task execution, which enables evolution to be guided by the search for novel or diverse behaviours instead of fitness.

.

The fitness function for a maze navigation task can, for instance, be defined based on how close the robot gets to the goal (Lehman and Stanley, 2011a). If a maze has no deceptive obstacles, this fitness function creates a monotonic gradient for the search to follow. However, mazes with obstacles that prevent a direct route may cause the fitness function to deceive evolution. If the candidate solution is instead characterised in behavioural terms, such as the robot's final position in the maze, searching for novel behaviours has the potential to avoid the deception associated with the fitness functions in a number of tasks.

Based on the insight that candidate solutions can be scored according to behaviour, Lehman and Stanley (2008, 2011a) introduced *novelty search*, in which the idea is to maximise the novelty of behaviours instead of their fitness, that is, to explicitly search for novel behaviours as a means to bootstrap evolution and to circumvent convergence to local optima. Specifically, the novelty search algorithm uses an archive to characterise the distribution of novel behaviours that are found throughout evolution. The algorithm operates by: (i) computing the *novelty score* of a behaviour by measuring the distance to the k-nearest neighbours, where k is a fixed parameter that is determined experimentally, and (ii) adding the behaviour to the archive stochastically or if it is significantly novel, that is, if the novelty score is above some minimal threshold. Because behaviours from more sparse regions of the behavioural search space receive higher novelty scores, the gradient of search is always towards what is novel, with no explicit objective. This new perspective on how to guide the evolutionary process triggered a significant body of work and added a new dimension not only to evolutionary robotics (Mouret and Doncieux, 2012; Gomes and Christensen, 2013; Pugh et al., 2015; Cully and Mouret, 2015; Mouret and Clune, 2015; Cully et al., 2015), but to evolutionary computation in general (Goldsby and Cheng, 2010; Naredo and Trujillo, 2013; Liapis et al., 2013).

Given its divergent nature, novelty search has been shown to be unaffected by premature convergence-related and bootstrapping issues than fitness-based evolution in a number of tasks, including maze navigation and biped locomotion with variable difficulty (Lehman and Stanley, 2011a; Lehman, 2012; Lehman and Stanley, 2011c; Lehman et al., 2013b), memory, cognitive learning, and communication tasks (Lehman and Miikkulainen, 2014;

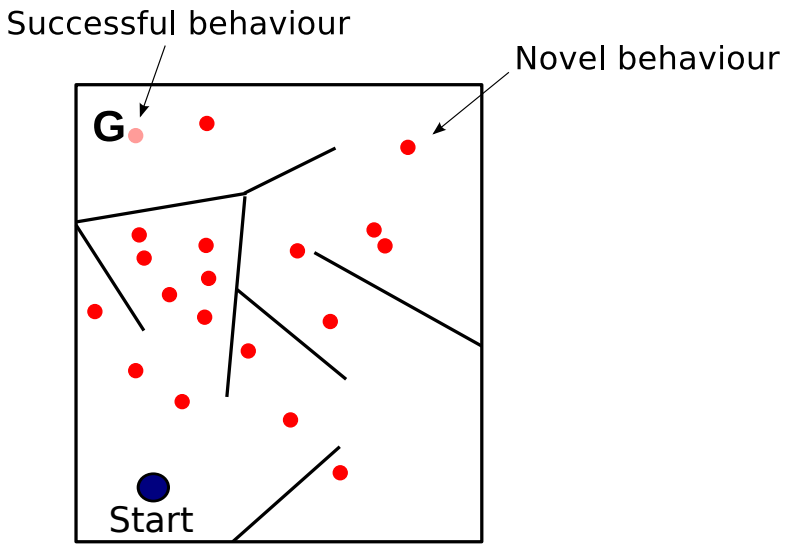


Figure 2.3: A maze with obstacles that prevent a direct route from the robot to the goal. The robot has to navigate from the start position (marked with "Start") to the goal (indicated by "G"). Red dots are examples of final positions (behaviour characterisations) of candidate solutions. Solutions that cause a robot to end up in sparse regions of the maze (the behaviour space) are considered novel.

.

Silva et al., 2015d), and collective robotics tasks such as resource sharing (Gomes and Christensen, 2013). Inspired by novelty search, different studies have introduced behavioural diversity-based methods that reward the novelty or the diversity of behaviours in different ways (Mouret and Doncieux, 2012). Novelty search and behavioural diversity methods are, however, significantly dependent on the *behaviour characterisation*, as shown by Kistemaker and Whiteson (2011), and can be challenging to apply when such a metric is not easy to define. That is, although the aforementioned methods operate independently of fitness, their effectiveness typically depends on a similar form of human knowledge. In addition, if the behaviour space is vast, novelty search and related methods may not scale well (Doncieux and Mouret, 2014).

Combining Fitness and Behavioural Diversity

In recent years, numerous approaches have been introduced to combine the respective advantages of fitness-based evolution and of diversity-based evolution, to obtain more effective optimisation procedures. One is the combination of the novelty score and of the fitness score using a weighted sum, as studied by Cuccu and Gomez (2011). Even though the study showed that combining the novelty and fitness scores may lead to better performance, the weights assigned to each score must be fine-tuned because different weightings can cause significant variations in the results (Cuccu and Gomez, 2011). A second approach is *minimal criteria novelty search* (MCNS) by Lehman and Stanley (2010). MCNS is an extension of novelty search in which individuals must meet one or more domain-dependent fitness criteria to be selected for reproduction (e.g. the fitness score of an individual must be at least $X\%$ of the theoretical maximum fitness score). The practical motivation behind MCNS is to reduce the size of behaviour spaces to enable a more tractable search for suitable behaviours. The results of Lehman and Stanley's MCNS study on two maze navigation tasks support the hypothesis that reducing the behaviour space during a search for novelty can often increase the efficiency of the search process. Although MCNS is an interesting method because it restricts the search for novelty to viable regions of the

behaviour space, its performance is contingent on a number of aspects, including: (i) choosing the minimal criteria should be done carefully because of the restrictions each criterion puts on the search space, and (ii) if no individuals are found that meet the minimal criteria, including in the initial population, there is no selection pressure and the evolutionary search enters a random drift (Lehman and Stanley, 2010).

An alternative approach is the Pareto-based multiobjective evolutionary algorithms that simultaneously optimise behavioural diversity and fitness (Mouret, 2011; Mouret and Doncieux, 2009, 2012). Contrarily to the weighted sum approach, which requires the weights assigned to each score to be fine-tuned, the multiobjective approach automates the exploration and exploitation phases based on the trade-offs between the behavioural diversity objective and the fitness objective. Throughout the search, fitness is maximised to the detriment of behavioural diversity at one extreme of the non-dominated Pareto front, while behavioural diversity is maximised at the expense of fitness at the opposite extreme. The continuum between the two extremes is therefore composed of multiple trade-offs between performance and diversity. Importantly, while behavioural diversity and fitness are often conflicting objectives, using both objectives enables the search to move in multiple non-dominated directions instead of in a single direction, which allows to more easily bypass local optima (Knowles et al., 2001). In effect, such multiobjective approaches can fine-tune behaviours more effectively than pure novelty search (Mouret, 2011), and increasingly outperform methods that rely, for instance, on novelty search or on fitness-based search alone as the task difficulty increases (Lehman et al., 2013b).

The more common multiobjective formulations combine behavioural diversity and fitness at a *global level*. A different formulation is *novelty search with local competition* (NSLC) (Lehman and Stanley, 2011b). In NSLC, the fitness objective is changed from being a global measure to being one relative to a controller's neighbourhood of *behaviourally similar* individuals. In practice, when the novelty score of a controller is computed, the number of nearest neighbours with lower fitness is also counted. This number is assigned as the *local competition* objective for that individual, which measures a controller's performance with respect to its behavioural niche. In this way, the two objectives become the novelty score and the local competitiveness score. In the context of evolution of robot body plans for virtual walking creatures, Lehman and Stanley (2011b) have shown that NSLC can effectively maintain and exploit a diversity of individuals at the expense of absolute performance.

The working principles of NSLC have originated a new class of approaches called *Quality Diversity* algorithms (Lehman and Stanley, 2011b; Cully and Mouret, 2013; Pugh et al., 2015, 2016) or *Illumination* algorithms (Mouret and Clune, 2015). The goal of Quality Diversity algorithms is to discover a wide range of diverse candidate solutions, but where each candidate solution can also be optimised for performance, that is, with respect to a measure of quality. One of the most adopted Quality Diversity algorithm is MAP-Elites (Mouret and Clune, 2015). Given a behaviour characterisation with N dimensions, MAP-elites first transforms the behaviour space into discrete bins according to a user-defined granularity level, and then tries to find the highest-performing individual for each point in the discretised space in order to construct a behaviour-performance map.

2.1.3 Early Pioneering Contributions

The possibility of evolving robots was first evoked by the neurophysiologist Valentino Braitenberg in his thought-experiment on the creation of new robot designs (Braitenberg, 1984). Almost a decade later, the field of evolutionary robotics began to develop. Pioneering groups at the University of Southern California, US (Lewis et al.,

1992), at the Swiss Federal Institute of Technology in Lausanne, Switzerland (Floreano and Mondada, 1994, 1996), at Sussex University in the UK (Harvey et al., 1994), and at the Italian National Research Council (Nolfi et al., 1994) laid the foundation for a number of important studies that followed.

Lewis et al. (1992) evolved neural network controllers for a real six-legged robot by synthesising the controllers on a workstation, and downloading each controller to the real robot for performance evaluation. Each evaluation required a human observer to monitor and score the performance of the real robot. Complementarily, Floreano and Mondada (1994, 1996) experimented with evolution of controllers directly in real robotic hardware, including navigation and homing behaviours for a Khepera robot. Given the low computational power of the Khepera robot, the actual evolutionary computation was performed on a workstation. The synthesis of successful controllers required up to ten days of continuous evolution. Harvey et al. (1994) evolved controllers for a real Gantry robot, and demonstrated principled approaches to the evolution of visually-guided robot behaviour (e.g. navigation and shape discrimination tasks), namely concurrent evolution of sensorimotor features and control systems. Finally, Nolfi et al. (1994) proposed evolving controllers in simulation and continuing evolution for a few generations in real hardware if a decrease in performance is observed when controllers are transferred.

The pioneering contributions facilitated progress and cross-fertilisation of ideas between different robotics domains. For example Brooks (1992), a pioneer in behaviour-based robotics, acknowledged the potential of evolutionary robotics techniques for control synthesis, and argued for different research avenues that are still being pursued, such as making evolution aware of regularities in morphological structure (e.g. symmetric sensor placement) and enabling to mirror them in the control structure (Silva et al., 2016d). In summary, early pioneering contributions highlighted the potential of evolutionary robotics and gave rise to a number of different research directions, see Silva et al. (2016a) for a review. At the current state of the art, the main axis of research in evolutionary robotics is arguably the engineering of control systems (Silva et al., 2016d). In this respect, researchers have been consistently faced with a number of issues, which we discuss in the following section.

2.1.4 Open Issues in Evolutionary Robot Engineering

After two and a half decades of research in evolutionary robotics, controllers have been successfully evolved for robots with varied functionality, from terrestrial robots to flying robots (Floreano et al., 2005). Although there has been significant progress in the field, it is arguably on a scale that still precludes the widespread adoption of evolutionary robotics techniques (Silva et al., 2014b). Evolved controllers are in most cases not yet competitive with human-designed solutions (Doncieux et al., 2011), and have only proven capable of solving relatively simple tasks such as obstacle avoidance, gait learning, and search tasks (Nelson et al., 2009). In effect, a number of critical issues currently prevent evolutionary robotics from becoming a viable mainstream approach for engineers. In our view, the most relevant issues are:

- The reality gap (Jakobi, 1997), which occurs when controllers evolved in simulation become ineffective once transferred to the physical robot due to their exploitation of features of the simulated world that are different or that do not exist at all in the real world. Differences between simulation and the real world can vary, from inaccurate sensor modelling to simulation-only artefacts due to simplifications, abstractions, idealisations, and discreteness of physics implementations.

- The prohibitively long time necessary to evolve controllers directly on real robots (Matarić and Cliff, 1996). Given its trial-and-error nature, evolutionary algorithms may require the evaluation of a large number of candidate solutions to the task. Combined with the fact that each evaluation can take a significant amount of time, the approach remains infeasible on real robots (Silva et al., 2016d).

- The bootstrap problem (Gomez and Miikkulainen, 1997; Nelson et al., 2009), which occurs when the task is too demanding for the fitness function to apply any meaningful selection pressure on a randomly generated population of initial candidate solutions: all individuals in the early stages of evolution may perform equally poorly, causing evolution to drift in an uninteresting region of the search space.

- Deception (Whitley, 1991), which occurs when the fitness function fails to build a gradient that leads to a global optimum, and instead drives evolution towards local optima. As a result, the evolutionary algorithm may converge prematurely to a sub-optimal solution, that is, it may *stagnate*.

- The design of genomic encodings and of the genotype-phenotype mappings that enable the evolution of complex behaviours (Meyer et al., 1998): the vast majority of evolutionary robotics studies employ *direct encoding* (Nelson et al., 2009), in which genotypes directly specify a phenotype. Even though direct encodings have been used with important results for controllers of relatively small size or with few parameters (Floreano et al., 2008), they are limited in their ability to evolve complex, large-scale controllers (Husbands et al., 1997; Meyer et al., 1998). Because each parameter of the controller has to be encoded and optimised separately, the size of the search space grows exponentially with the linear increase of controller size (Yao, 1999), which in turn leads to scalability issues. In effect, when evolutionary robotics emerged as a field of research, direct encodings were soon identified as one of the potential limitations for scaling evolutionary techniques to complex tasks (Husbands et al., 1997; Meyer et al., 1998).

- The absence of standard research practices in the field, including the lack of standard simulation platforms, benchmarks, and test-beds (Silva et al., 2016d).

Importantly, the differences between evolutionary robotics and more traditional domains mean that there is currently little theory and formal methods that can be applied to evolutionary robotics. Consider, for instance, the analysis of fitness landscapes. In traditional domains, multiple methods can be used to elucidate the properties of the search space and of fitness landscapes (Nelson et al., 2009), and to estimate task difficult in numerous scenarios (Poli et al., 2010). In evolutionary robotics, however, search spaces and corresponding fitness landscapes are challenging to characterise. Search spaces are generally rugged to an extreme degree, may have varying numbers of dimensions, and may potentially be non-static. In this way, full characterisation of search spaces and of fitness landscapes is often an intractable problem (Nelson et al., 2009).

2.2 Online Evolution of Behavioural Control

In this section, we review and discuss the state of the art in online evolution of behavioural control for robots. The section is divided into two subsections. First, we categorise the online evolutionary process with respect to how it is carried out, and where it is carried out. Afterwards, we review and classify contributions in the field.

2.2.1 Classifying the Online Evolutionary Process

Existing approaches to online evolution in robot systems fall into three categories (Eiben et al., 2010a): (i) distributed evolution, (ii) encapsulated evolution, and (iii) a hybrid approach, similar to an island model. In distributed evolution, each robot carries a single genotype. The evolutionary process takes place when robots meet and exchange genetic information. The main disadvantage of distributed evolution approaches is that the improvement of existing solutions is only based on the exchange of genetic information between robots. In large and open environments, where encounters may be rare or frequent transmission of information may be infeasible, the evolutionary process is thus prone to stagnation.

A complementary approach, encapsulated evolution, overcomes stagnation as each robot maintains a population of genotypes stored internally and runs a self-sufficient instance of the evolutionary algorithm locally. Alternative controllers are executed sequentially and their fitness is measured. Within this paradigm, robots adapt individually without exchanging genetic material with other robots. The main drawback of the approach is therefore the lack of transfer of knowledge or exchange of genetic information between robots, which can accelerate online evolution (Huijsman et al., 2011).

The two methodologies, distributed evolution and encapsulated evolution, can be combined, leading to a hybrid approach similar to an island model (Tanese, 1989). Each robot optimises the internal population of genomes through intra-island variation, and genetic information between two or more robots is exchanged through inter-island migration. The evolutionary process executing on each individual robot is therefore self-sufficient based on the internal population optimisation, and is potentially accelerated as a result of the inherent parallelism and exchange of information between robots.

Spatial perspective: onboard vs offboard

From the spatial perspective, it is furthermore possible to distinguish two cases: (i) the *onboard* case, where all necessary computation and the evolutionary operators such as selection, crossover, and mutation are executed by the robots themselves, considering their potentially limited processing power and storage capabilities, and (ii) the *offboard* case, where both the required computation and the evolutionary operators are performed with the aid of external equipment outside the robots. An external component could be, for instance, a computer interfaced with the robots in order to collect fitness information, manage the evolutionary process, and even carry out the behavioural control logic (receive sensory inputs from the robots, and compute the corresponding actuation values, such as the speed values for the wheels).

2.2.2 Background on Online Evolution of Control

As initially mentioned in Section 2.1.3, the first example of online evolution in a real, neural network-driven mobile robot was performed by Floreano and Mondada (1994, 1996). In their studies, the authors successfully evolved navigation and obstacle avoidance behaviours for a Khepera robot. Evolution was based on an online generational evolutionary algorithm, but with the actual computation being performed on a workstation due to the limitations of the robot hardware. The workstation orchestrated the evolutionary process and, at each evaluation, injected a new controller into the robot in order to assess its quality. The process was similar to a master-slave evolutionary algorithm, with the slave calculating fitness and the master orchestrating evolution.

The synthesis of successful controllers required up to ten days of continuous evolution on real robots, thus indicating that the evaluation time is a key aspect in real-robot experiments. Even the evolution of controllers for a standard navigation and obstacle avoidance task required approximately 2.71 days (population of 80 genomes, 100 generations, 39 minutes per generation).

In Mondada and Floreano (1995), the authors described the evolution of a grasping behaviour. Experiments conducted in Floreano and Mondada (1994) were extended by adding graspable balls to the environment and a gripper module to the robot. Synthesis of gripping behaviours was shown to be fairly complex because it involved object recognition and sensorimotor coordination, including positioning the body of the robot and the gripper in an adequate position. Similarly to the initial experiments, the evolutionary process required several days of continuous evolution, even after the gripper action was reduced to a fixed action pattern behaviour (McFarland and Bösser, 1993) in order to minimise the complexity of the task. Since online evolution proved to be prohibitively challenging, researchers have focused on the problems posed by evolving controllers directly on physical robots (Matarić and Cliff, 1996).

Evolution of Control for Multirobot Systems

> *Constructing tools from a collection of individuals is not a novel endeavour for man. A chain is a collection of links, a rake a collection of tines, and a broom a collection of bristles. Sweeping the sidewalk would certainly be difficult with a single or even a few bristles. Thus there must exist tasks that are easier to accomplish using a collection of robots, rather than just one.*
>
> *Kube and Zhang (1993)*

Multirobot systems provide numerous advantages such as the ability to solve more difficult tasks, robustness to the failure of individuals, parallelism of operation, and the possibility for division of labour strategies (Farinelli et al., 2004; Jones and Mataríc, 2006; Trianni, 2008). In online evolution, the fact that robots operate in parallel implies that they can also *learn* in parallel. Below, we describe current approaches for online evolution of behavioural control in multirobot systems. Approaches are categorised according to the classification scheme described in Section 2.2.1.

Distributed Evolution

The first attempt at truly autonomous continuous evolution in multirobot systems was performed by Watson et al. (1999, 2002), and entitled *embodied evolution*. The use of multirobot systems was motivated by the speed-up of evolution due to the inherent parallelism in groups of robots that evolve together in the task environment.

In embodied evolution, each robot maintains a virtual energy level, constrained by a minimum and maximum values, and a fitness score reflecting the individual performance. At each control cycle, robots update their individual fitness score depending on task performance, and then probabilistically broadcast a part of their stochastically mutated genome at a rate proportional to their fitness. If a robot broadcasts a gene string, the fitness score is decreased by a constant amount, a penalty in analogy to reproduction costs. Robots that receive gene transmissions incorporate this genetic material into their genome with a probability inversely proportional to their fitness. This way, selection and variation operators are implemented in a distributed manner through the interactions between robots.

Algorithm 1 Pseudo-code of the control program that implements the PGTA (Watson et al., 2002) and runs independently on every robot.

```
initialise_genes()
energy ← min_energy
loop
    if excited? then
        send(genes[random(num_genes)] + mutation)
    end if
    if received? AND receptive? then
        genes[indexof(received)] = valueof(received)
    end if
    do_task_specific_behaviour()
    energy ← limit(energy + reward − penalty)
end loop
```

The algorithm underlying embodied evolution is entitled Probabilistic Gene Transfer Algorithm (PGTA), a variation of the Microbial Genetic Algorithm (Harvey, 2011). In Algorithm 1, we show the pseudo-code of the control program that implements the PGTA. *indexof* and *valueof* return the locus and value of the received gene, respectively. *limit* bounds the energy value between the minimum and maximum energy levels. *random* returns an integer value in the range of its argument. The task specific behaviour includes reading sensor values, updating network outputs, setting motor speed/directions accordingly, and all behaviours related with the task itself such as monitoring performance and setting the values of *reward* (fitness update component) and *penalty* (gene string broadcast component, in analogy to reproduction costs).

Following Watson et al. (1999, 2002)'s publications on embodied evolution and the simplicity of the PGTA algorithm, a number of improvements and extensions were proposed. Bianco and Nolfi (2004) proposed a methodology involving a group of simulated robotic units with the ability to self-assemble to another robot. The simulated robots were simplified versions of the real s-bot robots (Mondada et al., 2004). While executing, each simulated robot was able to reproduce and grow by using the body of other robots. Reproduction was achieved by injecting the genome in another robot when in close proximity. Growth was due to physical self-assembly, which allowed robots to form *organisms* with a given morphology. The methodology was assessed in three sets of simulation-based experiments. Experimental results showed the emergence of morphologies with specific shapes well-adapted to different scenarios. However, a limiting aspect of the approach is the difficulty in carrying out similar experiments using groups of real robots (Trianni, 2006).

Wischmann et al. (2007) investigated the interplay of embodied evolution and lifelong individual learning in a simulated predator-prey scenario. Learning was implemented through isotropic-sequence-order learning using input correlations only (Porr et al., 2006), a form of reinforcement learning. The goal of the approach was to increase the adaptability of robots to novel environmental conditions. Each robot was given a maturation period during which no robot mating and no controller replacement could take place. The mechanism enabled robots to adapt using individual learning before being subject to any selection pressure. However, within the authors' experimental framework, the learning mechanisms considered did not yield any advantage.

Karafotias et al. (2011) proposed the Embodied Distributed Evolutionary Algorithm (EDEA), and assessed its performance in simulation. One of the objectives behind EDEA is trying to overcome the spatial coincidence necessary for the exchange of genetic material between robots. The algorithm encompasses a phase of *partner*

identification that attempts to give robots a global view of the population through long-range communication channels. Each robot contacts all robots in range, identifies potential partners for exchanging genetic material with, and then selects one candidate solution via a tournament selection of size 2. After choosing the candidate solution, the robot compares its own fitness with that of the prospective partner, and probabilistically applies recombination and mutation operators. The offspring is decoded and embodied as the new controller. EDEA was compared with an encapsulated approach called the $(\mu+1)$-online algorithm (see the following section for a description) in three sets of simulation-based experiments, and did not yield any meaningful advantage in terms of performance. In effect, EDEA is conceptually similar to a distributed approach introduced in Simões and Barone (2002). In such approach, robots execute a cyclic procedure composed of a working phase, in which they attempt to perform the task, and a synchronised mating phase that is triggered by an internal timer. In the mating phase, each robot uses a radio link to communicate its fitness score and genome to every other robot, while receiving the same information from the remaining robots.

Encapsulated Evolution

In the literature, approaches following the encapsulated evolution strategy are scarce when compared with distributed evolution or hybrid evolution approaches. Arguably, the most notable encapsulated evolution approaches were developed in the context of the SYMBRION project[3], which aimed at adaptation and evolution for so-called *symbiotic multirobot organisms* (Baele et al., 2009). Bredeche et al. (2009) proposed the (1+1)-online algorithm after the classic (1+1)-evolutionary strategy (de Jong, 2006). Montanier and Bredeche (2011) then extended the algorithm to incorporate a restart procedure as a means to potentially bypass local optima. In-between the two studies, Haasdijk et al. (2010) proposed the $(\mu+1)$-online algorithm, which employs a time-sharing mechanism where individuals are evaluated sequentially during a pre-defined evaluation period. With respect to Haasdijk et al.'s algorithm, subsequent studies examined: (i) distinct self-adaptive mechanisms for controlling the mutation operator (Eiben et al., 2010b) – results were not fully consistent across the tasks considered, yet suggested increased performance through a de-randomised self-adaptive mutation step size control mechanism, and (ii) the impact of a number of parameters on performance (Haasdijk et al., 2012).

Hybrid Evolution

Distinct approaches to hybrid online evolution have been proposed throughout the years. Prieto et al. (2010) proposed the real-time Asynchronous Situated Coevolution (r-ASiCo) algorithm. r-ASiCo is based on a reproduction mechanism in which individual robots carry an embryo. When robots meet, the embryo is genetically modified. If a controller is unable to solve the task, the embryo is used to produce a new controller. The approach was assessed in a real-robot collective cleaning task, in which robots had to pass over areas that accumulated virtual dirt over time in order to clean them. In terms of solution complexity, navigation patterns that caused robots to distribute approximately homogeneously in the environment, similarly to dispersion tasks, were sufficient to solve the task. Huijsman et al. (2011) introduced an approach based on the encapsulated $(\mu+1)$-online algorithm, and on a peer-to-peer evolutionary algorithm designed by Laredo et al. (2010). In a number of different simulation-based experiments, the hybrid method consistently yielded higher performance

[3]SYMBRION is an EU-funded FET project that started in January 2008, for the duration of five years, under grant agreement 216342. A complete description of SYMBRION is available at: `www.symbrion.eu`.

than both pure distributed evolution and pure encapsulated evolution. Similarly, García-Sánchez et al. (2012) investigated the effects of different migration policies in the exchange of genetic information between robots, also using an adaptation of the encapsulated $(\mu+1)$-online algorithm, and obtained results comparable to those of Huijsman et al. (2011).

Bredeche et al. (2012) introduced a variant of the EDEA algorithm (Karafotias et al., 2011) entitled *minimal* EDEA (mEDEA). Comparing with EDEA, mEDEA maintains a population of genomes during task execution, simpler communication routines between robots, and less complicated evolutionary operators. In Algorithm 2, we show the pseudo-code of the control program that implements the mEDEA algorithm. The overall idea of mEDEA is to consider the genomes as the atomic elements and the population of robots as a distributed substrate on which genomes "compete" with each other. The algorithm resembles the Selfish Gene metaphor (Dawkins, 1976): a given genome is "successful" if it spreads across the population, which implicitly requires minimising risk for its "vehicles", the robots, and maximising the number of mating/broadcast opportunities.

Each robot executing mEDEA maintains an energy level reflecting the individual task performance of the current controller. Controllers are assigned a fixed operation time, a *lifetime*. During task execution, robots continually broadcast their genomes to other robots in communication range as long as the energy level is above zero and the lifetime has not expired. If the energy level reaches zero during the lifetime of the controller, the robot will simply do not operate and will stand still. When a controller's lifetime expires, the algorithm randomly selects one of the received genomes, and mutates it. The genome is decoded into a new controller, the energy level is set to the default value, and the task execution and broadcasting operations restart. Within this setup, genomes that cause the robot to move more in the environment are more likely to spread at a higher rate than genomes that cause their host to stand still or move less.

Algorithm 2 Pseudo-code of the mEDEA algorithm.

```
initialise_genes()
energy ← default_energy_level
loop
    if genome_is_not_empty? then
        load(genome)
    end if
    for time = 0 to lifetime do
        if energy > 0? AND genome_is_not_empty? then
            do_task_specific_behaviour()
            energy ← updated_energy_level
            broadcast_genome()
        end if
        if received_genomes? then
            add_to_list_received_genomes(received_genomes)
        end if
    end for
    empty_genome()
    if is_not_empty(list_received_genomes) then
        genome ← apply_variation(select_random(list_received_genomes))
        energy ← default_energy_level
    end if
    list_received_genomes ← empty
end loop
```

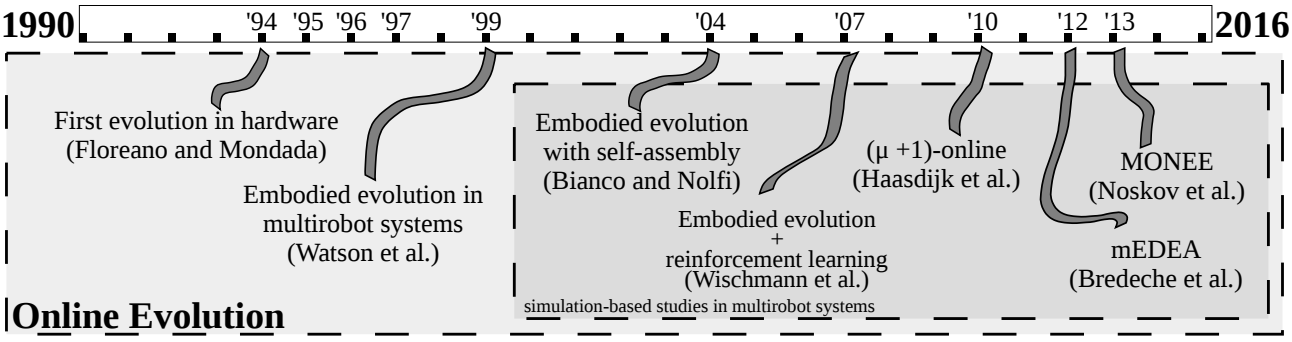


Figure 2.4: A chronogram of selected studies addressing online evolution. Although one of the goals of online evolution is to confine the evolutionary process to real robotic hardware, experiments reported in recent studies have been conducted mainly in simulation (simulated online evolution).

mEDEA is an interesting approach for implementing the so-called environment-driven online evolution but presents a number of shortcomings. Firstly, selection pressure in mEDEA is significantly dependent on the number of robots, and on the number of mating opportunities. The selection operator is limited to random sampling among the list of received genomes, that is, there is no selection pressure on a local individual basis. The most widespread genomes on a *global* population basis are more likely to be randomly selected *on average*. Moreover, if robots continuously receive the same number of copies of a given set of genomes (e.g. a group of robots is carrying a task such as box pushing, which requires them to be in close proximity), there is also no selection pressure on an individual basis. In this context, genome selection probabilities are uniform across the group, and the lack of selection pressure causes a random walk in genotypic space, that is, a random genetic drift. Secondly, mEDEA is prone to stagnation and sub-optimal performance. During the course of evolution, a number of robots may come to a halt either because: (i) they have not met any other robots, thus failing to import a new genome for use in the next iteration of the algorithm, or (ii) because the virtual energy level reached zero.

Despite the drawbacks of the algorithm, mEDEA was extended with a task-based selection scheme based on market mechanisms, an algorithm entitled MONEE (Noskov et al., 2013; Haasdijk and Bredeche, 2013). In essence, robots collect credits by performing tasks. If multiple tasks are defined, robots maintain separate counts of credits for each task. The count of credits is then used to perform a rank-based selection of the genomes received. The highest scoring genome is mutated and decoded into a new controller. The task-based selection scheme of MONEE alleviates the aforementioned lack of selection pressure in the original mEDEA, but does not approach the fact that the two algorithms are prone to stagnation and sub-optimal performance.

In summary, different approaches to online evolution have been introduced (see Figure 2.4). In effect, since embodied evolution (Watson et al., 1999, 2002) was developed, different tasks and types of robots have been used in online evolution studies, as shown in Table 2.1. Despite the progress, few studies on online evolution have been conducted on real robots (Nelson et al., 2009; Koos et al., 2013). Researchers have evaluated their proposed algorithms mainly through online evolution in simulation. As a result, even though new algorithms have been introduced and online evolution approaches have matured, the state of the art has not yet reached the point at which robots can adapt online in a timely manner (e.g. in one hour or less), and the class of tasks addressed has not increased in complexity (Silva et al., 2014b, 2015a).

Table 2.1: Summary of online evolution approaches introduced starting from the point embodied evolution was developed (Watson et al., 1999, 2002)

Approach	Tasks	Real robot
Embodied evolution (Watson et al., 1999, 2002)	Phototaxis	Yes
Embodied evolution with self-assembly (Bianco and Nolfi, 2004)	Open-ended survival	No
Embodied evolution + reinforcement learning (Wischmann et al., 2007)	Predator vs. prey pursuit	No
(μ + 1)-online (Haasdijk et al., 2010)	Navigation and obstacle avoidance	No
r-ASiCO (Prieto et al., 2010)	Collective cleaning	Yes
mEDEA (Bredeche et al., 2012)	Dynamic phototaxis	Partially
MONEE (Noskov et al., 2013)	Concurrent foraging	No

In this section, we reviewed the state of the art in online evolution of behavioural control for robots. Specifically, we described how the evolutionary process can be implemented in single and multirobot systems. In terms of the *type* of controller, a variety of models have been used in evolutionary robotics. According to Nelson et al. (2009), these include artificial neural networks, evolvable programs, various parametrised control structures, and evolvable hardware devices, among others. By 2009, artificial neural networks were already the predominant control system, and were estimated to have been used in approximately 40% of the studies (Nelson et al., 2009). Since then, the use of artificial networks has manifestly increased due to the development of more sophisticated neuroevolutionary methods (Stanley et al., 2009; D'Ambrosio et al., 2014; Silva et al., 2016d) that can, for example, correlate the internal geometry of the neural network with the placement of robot sensors and actuators, and then exploit task geometry for more effective problem solving (D'Ambrosio et al., 2014). In effect, all online evolution studies described in the previous section employed neural network-based control systems. The underlying online evolution algorithm, however, optimises artificial neural networks with a fixed, predefined topological structure. Fixed-topology methods require the system designer to decide on a suitable topology for a given task, which usually involves intensive experimentation. A non-optimal topology affects the evolutionary process and, consequently, the potential for adaptation. In the following section, we introduce fundamental concepts regarding artificial neural networks theory, and we review developments on fixed-topology and constructive neuroevolution algorithms.

2.3 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a computational processing paradigm inspired by the structure and function of biological nervous systems (Haykin, 1999). ANNs are able to approximate any continuous function

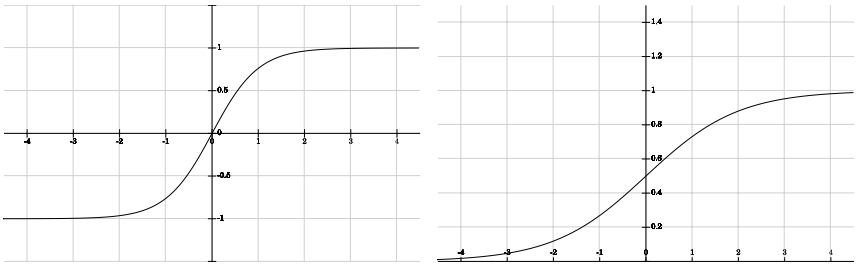
in theory (Cybenko, 1989), which makes them a relevant approach for control tasks. That is, ANNs have the potential to effectively control autonomous robots, vehicles, factories, or game players (Gomez and Miikkulainen, 2003; Stanley, 2004), among others.

2.3.1 Neurons and Activation Functions

A neural network consists of one or more interconnected processing units called nodes or *neurons*. A number of inputs $x_1, x_2, ..., x_N$ and one output y are associated to each neuron n in a network. Each input x_i has a weight w_i ($i \geq 1$), and w_0 specifies the weight of a bias input $x_0 = 1$. Each neuron n computes its activation value a_n, which is usually the weighted sum of the input values. a_n is then subject to an *activation function* f that computes the neuron's output value y, typically by mapping a_n to a value between 0 and 1.

For a neural network to compute non-linear functions, it is necessary to use non-linear activation functions (Haykin, 1999). In Figure 2.5, we show two commonly used non-linear activation functions: the hyperbolic tangent, and the logistic function (also called logistic curve). The two functions are sigmoid functions.[4]

$$f(a_n) = \frac{1}{1 + e^{-a_n}} \tag{2.1}$$



(a) (b)

Figure 2.5: Common non-linear neuron activation functions: (a) Hyperbolic tangent function, and (b) Logistic function.

2.3.2 Artificial Neural Network Architectures

In addition to one input layer and one output layer, ANNs can have one or more *hidden layers* of neurons. In effect, networks without hidden neurons can only compute functions that are linearly separable. An example of a network with hidden layers is shown in Figure 2.6. Each neuron in a hidden layer, a *hidden neuron*, receives the outputs of the neurons in the previous layer and then provides inputs to the neurons in the following layer.

A neural network where the information moves in only one direction and is propagated straightforward from the input neurons to the output neurons is called a *feedforward network*. While these networks are sufficient to calculate functions with static temporal behaviour, several problems require *memory*, that is, to recognise temporally extended patterns in input sequences – see, for instance, Ring (1994) and Lin et al. (1996). One way of implementing memory in neural networks is through *recurrent connections*, which are feedback connections to a given neuron, see Figure 2.7.

[4]While there are multiple sigmoid functions, the term *sigmoid function* is often used in the literature to describe the logistic function.

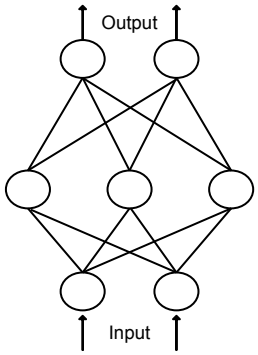


Figure 2.6: Example of a neural network with two input neurons, three hidden neurons, and two output neurons.

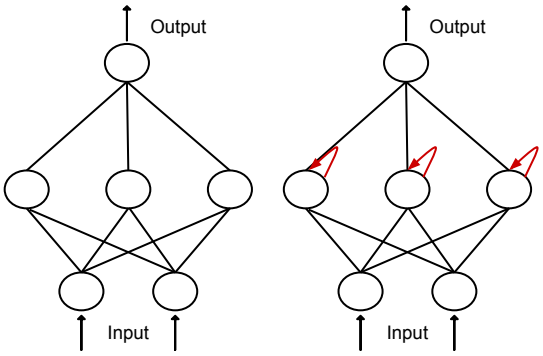


Figure 2.7: Neural network architectures. Black lines are feedforward, red lines represent recurrent connections. Left: A fully-connected single hidden layer feedforward network. Right: A recurrent network. Each hidden unit feeds the activation level back into itself.

For completeness, we describe a number of other approaches that can add memory capabilities to artificial neural networks. Architectures discussed so far are discrete time, and potentially recurrent, ANNs: each neuron is assumed to compute its output in one step. Another approach to implementing memory in discrete time neural networks is to introduce a type of neuron called *context unit* or *memory unit*, as in *Jordan networks* (Jordan, 1986), and *Elman networks* (Elman, 1990). The memory units and the context units maintain a copy of the previous values of the hidden neurons, that is, they are functionally equal to self-recurrent connections.

Additionally, more complex neural architectures are able to display intricate temporal properties. A neural network model able to explicitly deal with time-variant phenomena is the Continuous-Time Recurrent Neural Network (CTRNN) (Funahashi and Nakamura, 1993; Yamauchi and Beer, 1994). Each neuron in a CTRNN has a temporal response relating its state to the inputs. The temporal response is described by a differential equation. For each neuron n, the rate at which both incoming activation is integrated and existing internal activation is leaked is controlled by a time constant τ_n. The time constants allow different neurons to display distinct temporal properties. Large time constants result in slowly-changing neuron states, while small time constants approximate reactive neurons (Blynel and Floreano, 2002). An approach conceptually similar to CTRNNs is spiking neuron-based ANNs (Maass, 1997; Gerstner and Kistler, 2002). Compared with CTRNNs, spiking neural networks can potentially solve tasks with less structure due to their time-coding capabilities (Saggie et al., 2004). However, in computational terms, spiking neurons are significantly more expensive than the other ANN models described.

2.4 Neuroevolution

In order to use ANNs as robot controllers, it is necessary to determine the type of neural network to use, the topology, and the weighting parameters. One effective approach to neural network optimisation is *neuroevolution* (Floreano et al., 2008). In neuroevolution, the parameters of the ANN such as the connection weights, the neuron bias terms, and occasionally the topological structure, are optimised by an evolutionary algorithm. ANNs are well-suited for optimisation through evolution because small changes in the network typically correspond to small changes in its behaviour. That is, ANNs provide evolution with a relatively smooth space, thus allowing the evolutionary algorithms to gradually progress towards a solution (Floreano and Mondada, 1994).

The idea of using evolutionary algorithms to automate the design of neural networks dates back, at least, to 1989 (Harp et al., 1989; Kitano, 1990; Boers and Kuiper, 1992; Gruau, 1992; Angeline et al., 1994). Since then, neuroevolution has been successfully applied to control tasks in distinct domains (Yao, 1999; Stanley and Miikkulainen, 2002; Stanley, 2004; Gomez and Miikkulainen, 2003). In this section, we review important developments on fixed-topology and constructive neuroevolution algorithms. Our review focuses on direct encoding-based algorithms, which is the encoding scheme used in our online evolution research. For a discussion on the role of the genomic encoding in the evolution of robot controllers, see Appendix A.

2.4.1 Fixed-Topology Neuroevolution Algorithms

Early work on neuroevolution optimised weights for fixed-topology networks, driven by the theoretical results showing that a neural network with a single hidden layer could approximate any function, given enough neurons (Hornik et al., 1989). Below, we describe three relevant fixed-topology approaches in the literature.

SANE: Symbiotic, Adaptive Neuroevolution

Symbiotic, Adaptive NeuroEvolution (SANE) (Moriarty and Miikkulainen, 1996) takes an interesting approach to neuroevolution. The algorithm evolves a population of neurons instead of evolving a population of networks. Each "neuron genome" encodes the input and output connection weights of a neuron. Each population member therefore represents only a partial solution to the task. During the evaluation stage, random neurons are selected from the population, and combined to form the hidden layer of a feedforward network. Each neuron receives the average fitness of all the networks in which it was included. Multiple neurons are thus evaluated at the same time and rewarded for their generality. By evolving individual neurons to cooperate in networks, SANE automatically maintains diversity in the neuron population. Since different types of neurons are usually necessary to solve a problem, networks with too many copies of the same neuron are likely to fail. This way, SANE implicitly encourages the emergence of specialised sub-populations (Moriarty and Miikkulainen, 1996).

ESP: Enforced Sub-Populations

Enforced Sub-Populations[5] (ESP) (Gomez and Miikkulainen, 1997) is an improvement of SANE that explicitly divides an evolving population into separate sub-populations, one for each neuron in the evolving topology. Species do not self-organise since they are enforced from the start in an explicit niching scheme. Members

[5]In fact, the operation principle of ESP is similar to CCGA (de Jong and Potter, 1995), a cooperative coevolution algorithm that evolves specialised partial solutions (e.g. rules for a rule-based control system) on a set of sub-populations. The members of a given sub-population are not recombined with members from other sub-populations.

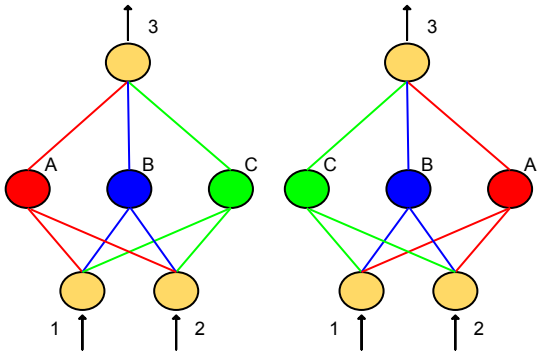


Figure 2.8: The Variable Length Genome Problem. Both networks compute the same exact function even though their hidden units appear in a different order. These are two of the six possible permutations of hidden units. Since the same neurons occupy different positions in the two networks, it is difficult to combine them without losing functionality.

from each population are combined together to form a complete network, which is then evaluated on the target task. Performance credit is divided between the neurons that contributed to the network. Since recombination only occurs between neurons of the same sub-population, each sub-population is forced to specialise into a sub-function for the network as a whole. Additionally, ESP is able to evolve recurrent connections in ANNs, whereas SANE is not.

CMA-ES: Covariance Matrix Adaptation Evolution Strategy

A distinct approach is the adaptation of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), originally proposed in Hansen and Ostermeier (2001), to the evolution of fixed-topology ANNs (Igel, 2003). CMA-ES keeps track of correlations between changes in network weights and fitness scores, and updates the covariance matrix of the weight mutation distribution so that it becomes more biased towards the most promising directions of search. CMA-ES has proven effective on distinct benchmark problems (Igel, 2003). The main drawback of the approach is the high computational cost and space-time complexity, which makes it unsuitable for large-scale optimisation problems with hundreds or thousands of dimensions (Omidvar and Li, 2011).

2.4.2 Constructive Neuroevolution Algorithms

In fixed-topology neuroevolution algorithms, the experimenter has to decide on a suitable topology for a given task, which usually involves a substantial amount of experimentation and human knowledge. Choosing an inappropriate topology affects the evolutionary process because: (i) networks too large have extra weights, and each of these adds an extra dimension to the search space, and (ii) networks too small may be unable to represent solutions beyond a certain level of complexity, which potentially limits their performance.

Contrarily to fixed-topology algorithms, *constructive* algorithms are able to simultaneously optimise both the weights and the network topology of ANNs. One challenging issue faced by constructive algorithms is how to efficiently explore the space of network topologies, that is, how to progressively augment neural networks without radically increasing the number of parameters to optimise (Stanley and Miikkulainen, 2002). A distinct issue is the *Competing Conventions*, also known as the *Variable Length Genome Problem* in constructive algorithms, see Figure 2.8. The main idea is that there is typically more than one way to express a solution to an ANN weight

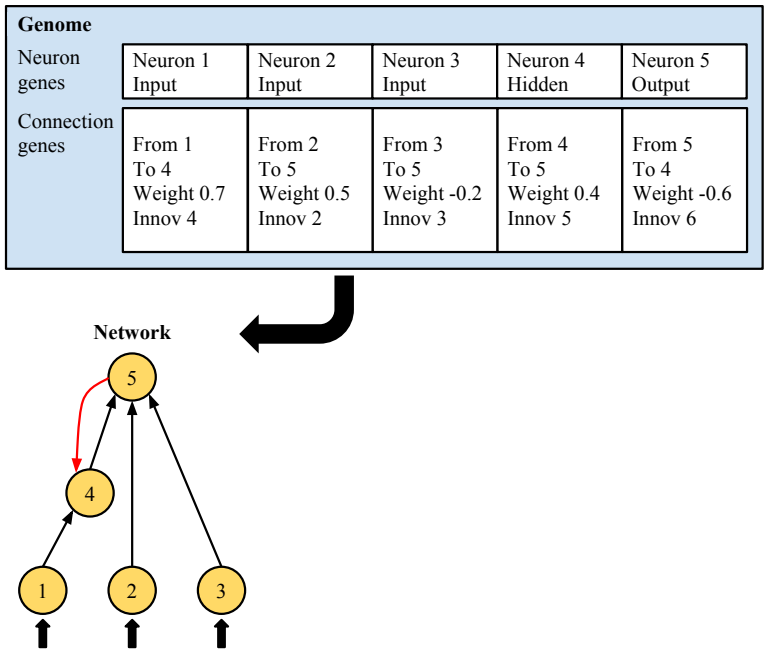


Figure 2.9: A genotype to phenotype mapping example.

optimisation task. In effect, there are potentially numerous symmetric solutions. When genomes representing networks that compute the same function do not have a matching representation, the application of a crossover operator will typically produce invalid offspring (Stanley and Miikkulainen, 2002). In this way, there are constructive neuroevolution algorithms that omit crossover and rely on mutation alone (Kassahun and Sommer, 2005; Siebel et al., 2007).

In the following section, we introduce an algorithm called NEAT (Stanley and Miikkulainen, 2002) that, among other relevant features, tackles variable-length genomes in an effective way. NEAT tracks genes (neurons and connections) through evolution via *historical markers* and is able to recognise which genes match up. In this way, issues related with competing conventions are alleviated because genes can be effectively aligned before recombination is carried out.

NeuroEvolution of Augmenting Topologies (NEAT)

The NeuroEvolution of Augmenting Topologies (NEAT) method (Stanley and Miikkulainen, 2002) is one of the most prominent offline neuroevolution algorithms. NEAT optimises both network topologies and weighting parameters. NEAT executes with global and centralised information like canonical evolutionary algorithms, and has been successfully applied to distinct problems, outperforming several methods that use fixed-topologies (Stanley and Miikkulainen, 2002; Stanley, 2004). The high performance of NEAT is due to three key features (Stanley and Miikkulainen, 2002; Stanley, 2004; Stanley et al., 2005): (i) the tracking of genes with *historical markers* to enable meaningful crossover between networks with different topologies, (ii) a *niching scheme* that protects topological innovations, and (iii) the incremental evolution of topologies from simple initial structures, that is, *complexification*.

In NEAT, the network connectivity is represented through a flexible genetic encoding (see Figure 2.9). Each genome contains a list of neuron genes and a list of connection genes. Connection genes encompass:

(i) references to two neuron genes, which respectively represent the two neurons being connected, (ii) the weight of the connection gene, (iii) one bit indicating if the connection gene should be expressed or not, and (iv) a *global innovation number*, unique for each gene in the population. Innovation numbers are assigned sequentially, and they therefore represent a chronology of the genes introduced. Genes that express the same feature are called *matching* genes. Genes that do not match are either *disjoint* or *excess*, depending on whether they occur within or outside the range of the other parent's innovation numbers. When crossover is performed (see Figure 2.10), matching genes are aligned, and inherited from any parent randomly. Genes that do not match are inherited from the fittest parent or, if they are equally fit, from any parent randomly. The NP-hard problem of matching

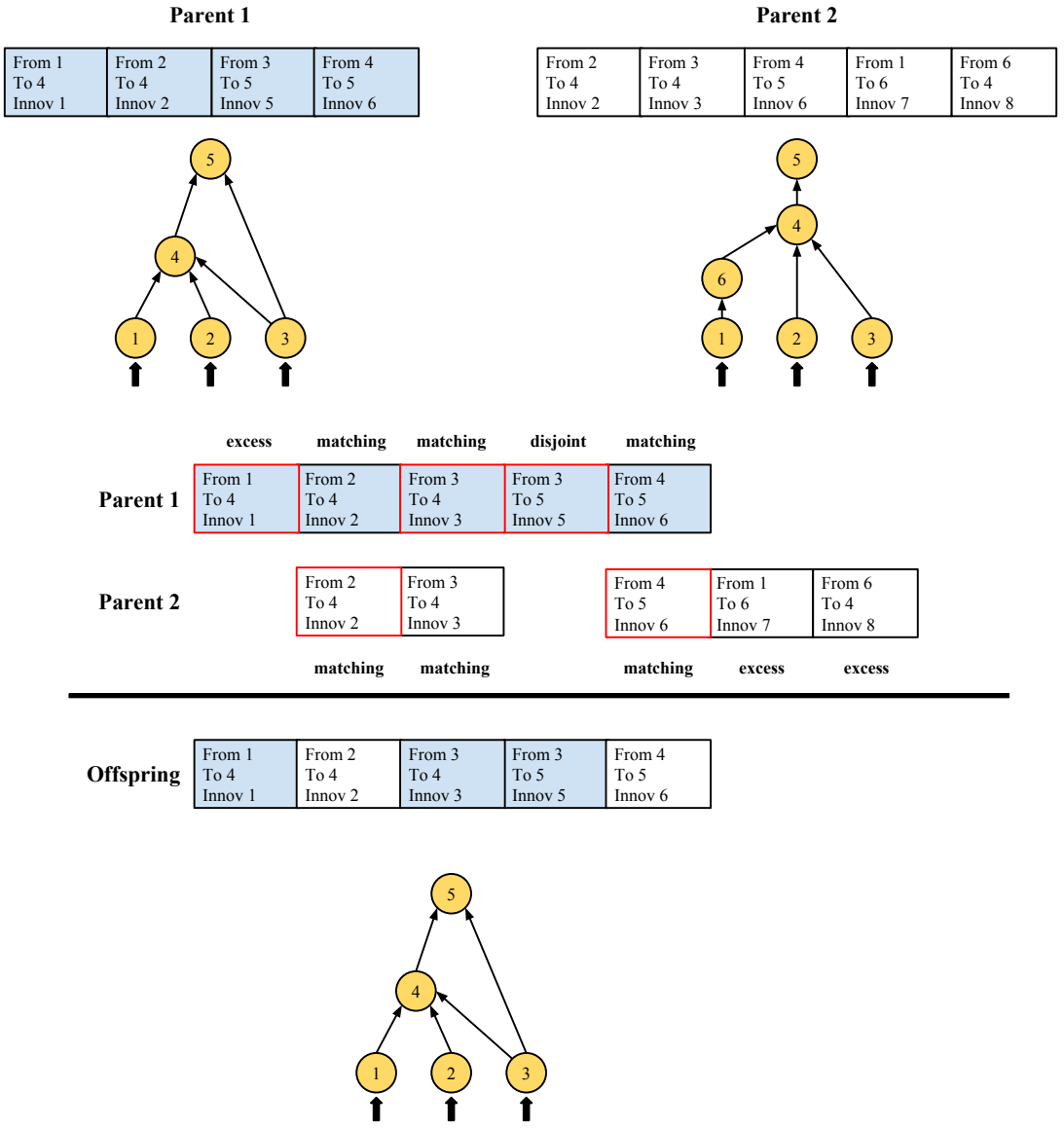


Figure 2.10: Matching up genomes for different network topologies using innovation numbers. Although Parent 1 and Parent 2 are different, their innovation numbers indicate which genes match up without the need for topological analysis. Parent 1 is fitter than Parent 2. Genes selected to form the offspring are marked in red. Because excess and disjoint genes are inherited from the fittest parent, neuron 6 of Parent 2 is not present in the offspring.

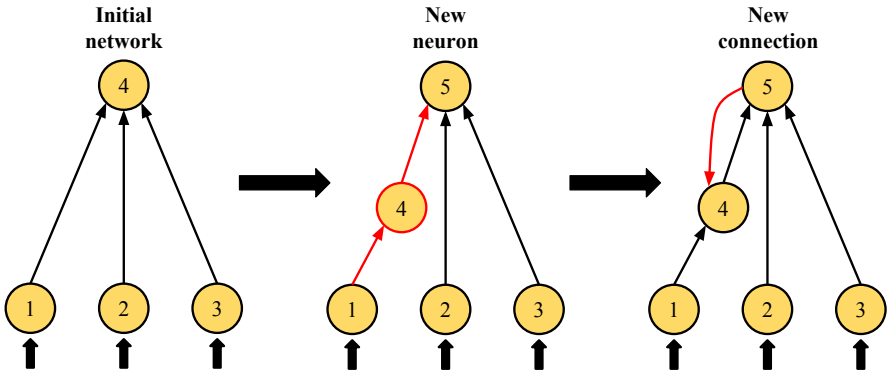


Figure 2.11: Complexification policy in NEAT. A new neuron and a new connection are added throughout evolution, which allows the network to grow in an incremental manner.

distinct network topologies is thus avoided and crossover can be performed without *a priori* topological analysis. In terms of mutations, NEAT allows for classic connection weight and neuron bias perturbations, and structural changes that lead to the insertion of either a new connection between two previously unconnected neurons, or a new neuron. A new neuron gene, representing the new neuron in the ANN, is introduced in the genome by splitting an old connection gene into two new connection genes.

NEAT protects new structural innovations by reducing competition between genomes representing differing structures and network complexities. The niching scheme is composed of speciation and fitness sharing. Speciation divides the population into non-overlapping sets of similar genomes based on the amount of evolutionary history they share. The compatibility between genomes is computed as a linear combination of the number of excess (E) and disjoint genes (D), and the average weight difference of matching connection genes ($\overline{W}$):

$$\delta = \frac{c_1 \cdot E}{N} + \frac{c_2 \cdot D}{N} + c_3 \cdot \overline{W} \tag{2.2}$$

The coefficients c_1, c_2, and c_3 determine the relative importance of the three factors. N is the number of genes in the larger genome and normalises for genome size. The compatibility measure is used to speciate with respect to a compatibility threshold δ_t. If a genome's distance to a randomly chosen member of the species is less than δ_t then the genome is placed into that species.

Fitness sharing dictates that individuals in the same species share the fitness of their niche. The fitness scores of members of a species are first *adjusted*, that is, divided by the number of individuals in the species. Species then grow or shrink in size depending on whether their average adjusted fitness is above or below the population average. Since the size of the species is taken into account in the computation of the adjusted fitness, new smaller species are not discarded prematurely, and one species does not dominate the entire population.

NEAT performs an incremental exploration of the search space through *complexification*, see Figure 2.11. The algorithm starts with a uniform population of simple networks with no hidden neurons, and with each input neuron connected to every output neuron. New neurons and new connections are then progressively added to the networks as a result of structural mutations. Because NEAT speciates the population, the algorithm effectively maintains a variety of networks with different structures and different complexities over the course of evolution. In this way, NEAT can search for an appropriate degree of complexity to the current task. Additionally, variants

of NEAT such as FS-NEAT (Whiteson et al., 2005) and Modular NEAT (Reisinger et al., 2004) use feature selection, allowing for input and output connections to be added during evolution in order to potentially find the set of inputs that may lead to better performance.

rtNEAT: Real-time NEAT

Real-time NeuroEvolution of Augmenting Topologies (rtNEAT) was introduced by Stanley et al. (2005) with the purpose of evolving ANNs *online*. rtNEAT is a centralised real-time version of NEAT originally designed for video games. Compared with NEAT, rtNEAT differs in a number of aspects, namely: (i) rtNEAT is a steady-state algorithm, while NEAT is generational, (ii) rtNEAT produces one offspring at regular intervals, every n time steps, and (iii) unlike NEAT, in which the number of species may vary, rtNEAT attempts to keep the number of species constant. To that end, rtNEAT adjusts a threshold C_t that determines the degree of topological similarity necessary for individuals to belong to a species. When there are too many species, C_t is increased to make species more inclusive; when there are too few, C_t is decreased to make species less inclusive. Despite these differences, rtNEAT has shown to preserve the dynamics of NEAT, namely complexification and protection of innovation through speciation (Stanley et al., 2005).

Even though rtNEAT only creates one offspring at a time, it approximates NEAT's behaviour, in which a number of offspring n_k is assigned to each species. In rtNEAT, a parent species S_k is chosen proportionally to its average adjusted fitness, as follows:

$$Pr(S_k) = \frac{\overline{F}_k}{\overline{F}_{total}} \tag{2.3}$$

where $\overline{F}_k$ is the average adjusted fitness of species k, and $\overline{F}_{total}$ is the sum of all species' average adjusted fitness. As a result, the expected number of offspring for each species in rtNEAT is, over time, proportional to n_k in traditional NEAT. The remaining process of reproduction is similar to NEAT's: two parents are chosen, and crossover and mutation operators are applied probabilistically. The individual with the lowest adjusted fitness is then removed from the population and replaced with the new offspring.

2.5 Summary

In this chapter, we provided an introduction to the field of evolutionary robotics, reviewed and discussed the state of the art in online evolution, and described various types of artificial neural networks. Finally, we discussed influential approaches to neuroevolution, with particular emphasis on the constructive neuroevolution algorithm NEAT (Stanley and Miikkulainen, 2002). NEAT has proven to be effective in a number of different domains. For instance, NEAT has outperformed ESP and other methods in complex control tasks such as non-Markovian double pole balancing (Stanley and Miikkulainen, 2002). In particular, the complexification methodology used by NEAT has shown to be able to find a higher level of behavioural sophistication than fixed-topology algorithms on robotic strategy-learning (Stanley and Miikkulainen, 2004). Other successful applications of NEAT are described in Stanley (2004) and include: (i) a roving eye[6] for the Go board game,

[6] A roving eye is a general concept in machine vision, referring to a visual field smaller than the total relevant image area; such a field must move around the image in order to process its entire contents. Roving eyes are often used in robots, where they allow successful navigation even with a limited sensory radius. This type of purposeful control of a moving visual field is also sometimes called *active vision*.

(ii) teaching a simulated car to predict crashes in an automobile warning system, and (iii) real-time evolution of game characters while they are playing against humans (using rtNEAT). In the following chapter, we extensively assess an approach to online evolution of controllers in multirobot systems called odNEAT (Silva et al., 2015e) that is inspired by NEAT.

Chapter 3

Online Decentralised NeuroEvolution of Augmenting Topologies

This chapter offers a comprehensive analysis of Online Decentralised NeuroEvolution of Augmenting Topologies (odNEAT), initially introduced in Silva et al. (2012). odNEAT is an algorithm for online evolution of artificial neural network (ANN) controllers in multirobot systems. odNEAT is distributed across multiple robots that evolve in parallel and exchange candidate solutions to the task when they meet. odNEAT differs from more traditional approaches to online evolution because both the weighting parameters and the topology of ANNs are under evolutionary control. In previous contributions, online approaches have been typically limited to the evolution of weighting parameters in fixed-topology ANNs. Fixed-topology methods require the system designer to decide on a suitable topology for a given task, which usually involves intensive experimentation. A non-optimal topology affects the evolutionary process and, consequently, the potential for adaptation.

To the best of our knowledge, only one other online method that optimises neural topologies and weights in a decentralised manner has previously been proposed; the IM-$(\mu + 1)$ algorithm (Schwarzer et al., 2011). In addition, the goal of the centralised rtNEAT algorithm (Stanley et al., 2005) in video games is conceptually similar to that of odNEAT. rtNEAT was developed to allow non-player characters to optimise their behaviour during a game. We therefore set up a series of simulation-based experiments in which we compare the performance of odNEAT with the performance of rtNEAT and IM-$(\mu + 1)$.

The chapter is organised as follows. In Section 3.1, we introduce the IM-$(\mu + 1)$ algorithm. In Section 3.2, we present and motivate the features of odNEAT. In Section 3.3, we explain our experimental methodology and the three tasks used in this study. In Section 3.4, we present and discuss our experimental results. Section 3.5 is dedicated to the exploration and analysis of odNEAT in terms of long-term self-adaptation, fault tolerance, and the influence of each algorithmic component on performance. In Section 3.6, we summarise the chapter and provide concluding remarks.

3.1 IM-$(\mu + 1)$ algorithm

The IM-$(\mu+1)$ algorithm (Schwarzer et al., 2011) is a variant of the $(\mu+1)$-online algorithm, proposed by Haasdijk et al. (2010), with random ad-hoc exchange of genomes between robots. Below, we describe the similarities between IM-$(\mu + 1)$ and rtNEAT, and then the execution loop of IM-$(\mu + 1)$.

The IM-$(\mu+1)$ algorithm mimics a number of features of rtNEAT. Firstly, new genes are assigned innovation

numbers. An important difference with respect to rtNEAT is that in IM-($\mu + 1$), innovation numbers are local and randomly chosen from a predefined interval to enable a decentralised implementation. The interval $[1, 1000]$ was used in Schwarzer et al. (2011). Secondly, as in rtNEAT, crossover is performed between similar genomes to maximise the chance of producing meaningful offspring. Explicit speciation is not used in IM-($\mu + 1$). Instead, genomes that have a high degree of *genetic similarity* are recombined. Genetic similarity is computed based on the ratios of matching connection genes and matching neuron genes.

Thirdly, IM-($\mu + 1$) follows rtNEAT's complexification policy and usually starts evolution from simple topologies, although it can be seeded with a parametrisable number of hidden neurons (Schwarzer et al., 2011). IM-($\mu+1$) allows for non-structural mutations similarly to rtNEAT, namely connection weight and neuron bias perturbations, but it employs different methods for optimising the topology of the ANN. Structural mutation of a connection gene can either remove the gene or introduce a new connection gene in a random location and with a randomly assigned weight. In the same way, structural mutation of a neuron gene can either delete the gene or produce a new neuron gene with a random innovation number and a random bias value. In addition, insertion of new neuron genes and connection genes can be performed through duplication and differentiation of an existing neuron gene, and of its incoming and outgoing connection genes.

3.1.1 Execution Loop

In the IM-($\mu+1$) algorithm, as in the encapsulated ($\mu+1$)-online, each robot maintains an internal population of μ genomes. Robots in close proximity can exchange genomes according to a migration policy that presupposes bilateral and synchronous communication. When two robots meet, they exchange genomes if none of them has had a migration within a predefined period of time. Each robot randomly selects and *removes* one genome from its internal population, and probabilistically applies crossover and mutation. The resulting genome is transmitted to the neighbouring robot, and the genome received is incorporated in the internal population.

During task execution, $\lambda = 1$ new offspring is produced at regular time intervals. One genome from the population of μ genomes maintained by the robot is randomly selected to be a parent. A second parent is used if there is any genome in the population considered genetically similar. The offspring resulting from crossover and mutation is decoded into an ANN that controls the robot. The controller operates for a fixed amount time called the *evaluation period*. When the evaluation period elapses, the evaluated genome is added to the population of the robot. The genome with the lowest fitness score is subsequently removed to keep the population size constant.

3.2 Introducing odNEAT

In this section, we present the odNEAT algorithm, in which both the weighting parameters and the topology of the ANNs are under evolutionary control. One of the motivations behind odNEAT is that, by evolving neural topologies, the algorithm bypasses the inherent limitations of fixed-topology online algorithms. As in rtNEAT, odNEAT starts with simple controllers, in which the input neurons are connected to the output neurons. The ANNs are progressively augmented with new neurons and new connections, and a suitable network topology is the product of a continuous evolutionary process.

odNEAT adopts rtNEAT's matching and recombination of neural topologies, and niching scheme. The

differences between odNEAT, rtNEAT, and IM-($\mu + 1$) are summarised in Table 3.1. odNEAT differs from rtNEAT and IM-($\mu + 1$) in a number of key aspects, namely:

- The internal population of each robot is constructed in an incremental manner.
- odNEAT maintains a local *tabu list* of recent poor solutions, which enables a robot to filter out genomes representing controllers similar to those that failed recently.
- odNEAT follows a unilateral migration policy for the exchange of genomes between robots. Copies of the genome encoding the active controller are probabilistically transmitted to nearby robots if they have a competitive fitness score and represent topological innovations with respect to the robot's internal population of solutions.
- odNEAT uses local, high-resolution timestamps. Each robot is responsible for assigning a timestamp to each local innovation, be it a new connection gene or a new neuron gene. The use of high-resolution timestamps for labels practically guarantees uniqueness and allows odNEAT to retain the concept of chronology.
- A controller remains active as long as it is able to solve the task. A new controller is only synthesised if the current one fails, that is, when it is actually necessary.
- New controllers are given a minimum amount of time controlling the robot, a *maturation period*.

Table 3.1: Summary of the main differences between odNEAT, rtNEAT, and IM-($\mu + 1$)

	odNEAT	rtNEAT	IM-($\mu + 1$)
Population	Distributed	Centralised	Distributed
Niching scheme	Yes	Yes	No
Tabu list	Yes	No	No
Migration policy	Unilateral	n/a	Synchronous
Innovation numbers	High-res. timestamps	Sequential	Random
Controller replacement	When previous fails	Periodically	Periodically
Maturation period	Yes	No	No

3.2.1 Virtual Energy Level and Fitness Score

In odNEAT, robots maintain a virtual energy level reflecting the individual task performance of the current controller. Controllers are assigned a default and domain-dependent virtual energy level when they start executing. The virtual energy level increases and decreases as a result of the robot's behaviour. If a controller's virtual energy level reaches a minimum threshold, a new controller is produced.

One common issue, especially for highly complex tasks, is that online evaluation is inherently noisy. Very dissimilar conditions may be presented to different genomes when they are translated into a controller and evaluated. The location of a robot within the environment and the proximity to other robots, for instance, are factors that may have a significant influence on performance and behaviour. With the purpose of obtaining a more reliable fitness estimate, odNEAT distinguishes between the fitness score of a solution and its current virtual energy level. The fitness score is defined as the average of the virtual energy level, sampled at regular time intervals.

3.2.2 Internal Population, Exchange of Genomes, and Tabu List

In odNEAT, each robot maintains a local set of genomes in an internal population. The internal population is subject to speciation and fitness sharing. The population contains genomes generated by the robot, namely the current genome and previously active genomes that have competitive fitness scores, and genomes received from other robots. Every control cycle, a robot probabilistically broadcasts a copy of its active genome and the virtual energy level of the corresponding controller to robots in its immediate neighbourhood, an *inter-robot reproduction event*, with a probability computed as follows:

$$P(event) = \frac{\overline{F}_k}{\overline{F}_{total}} \tag{3.1}$$

where $\overline{F}_k$ is the average adjusted fitness of local species k to which the genome belongs and $\overline{F}_{total}$ is the sum of all local species' average adjusted fitnesses. The broadcast probability promotes the propagation of topological innovations with a competitive fitness.

The internal population of each robot is updated according to two principles. Firstly, a robot's population does not allow for multiple copies of the same genome. Due to the exchange of genetic information between the robots, a robot may receive a copy of a genome that is already present in its population. Whenever a robot receives a copy C' of a genome C, the virtual energy level of the controller that corresponds to C' is used to incrementally average the fitness of C, and thus to provide the receiving robot with a more reliable estimate of the genome's fitness. Secondly, whenever the internal population is full, the insertion of a new genome is accompanied by the removal of the genome with the lowest adjusted fitness score. If a genome is removed or added, the corresponding species has either increased or decreased in size, and the adjusted fitness of the species is recalculated.

Besides the internal population, each robot also maintains a local tabu list, a short-term memory which keeps track of recent poor solutions, namely: (i) genomes removed from the internal population because it got full, or (ii) genomes whose virtual energy level reached the minimum threshold, and therefore failed to solve the task. In the latter case, the genome is added to the tabu list but it is maintained in the internal population as long as its fitness is comparatively competitive.

The tabu list filters out solutions broadcast by other robots that are similar to those that have already failed. The purpose of the tabu list is: (i) to avoid flooding the internal population with poor solutions, and (ii) to keep the evolutionary process from cycling around in an unfruitful neighbourhood of the solution space. Received genomes must first be checked against the tabu list before they become part of the internal population. Received genomes are only included in the internal population if they are topologically dissimilar from all genomes in the tabu list.

3.2.3 New Genomes and the Maturation Period

When the virtual energy level of a controller reaches the minimum threshold, because it is incapable of solving the task, a new genome is created. In this process, *an intra-robot reproduction event*, a parent species is chosen proportionally to its average adjusted fitness, as defined in Equation (3.1). Two parents are selected from the species, each one via a tournament selection of size 2. The offspring is created based on crossover of the

parents' genomes and mutation of the new genome. Once the new genome is decoded into a new controller, it is guaranteed a maturation period during which no controller replacement takes place. The new controller can continue to execute after the maturation period if its energy level is above the threshold.

odNEAT's maturation period is conceptually similar to the recovery period of the $(1 + 1)$-online algorithm (Bredeche et al., 2009), and its purpose is twofold. Firstly, the maturation period is important because the new controller may be an acceptable solution to the task but the environmental conditions in which it starts to execute may be unfavourable. Thus, the new controller is *protected* for a minimum period of time. Secondly, the maturation period defines a lower bound of activity in the environment that should be sufficiently long to enable a proper estimate of the quality of the controller. This implies a trade-off between a minimum evaluation time and how many candidate solutions can be evaluated within a given amount of time. A longer maturation period increases the reliability of the fitness estimate of solutions that fail, that is, of the stepping stones of the evolutionary process, while a shorter maturation period increases the number of solutions that can be evaluated within a given period of time. In Algorithm 3, we summarise odNEAT as executed by each robot.

Algorithm 3 Pseudo-code of odNEAT that runs independently on every robot.

```
genome ← create_random_genome()
controller ← assign_as_controller(genome)
energy ← default_initial_energy
loop
    if broadcast? then
        send(genome, robots_in_communication_range)
    end if
    if has_received? then
        for all c in received_genomes do
            if tabu_list_approves(c) and population_accepts(c) then
                add_to_population(c)
                adjust_population_size()
                adjust_species_fitness()
            end if
        end for
    end if
    operate_in_environment()
    energy ← update_energy_level()
    if energy ≤ minimum_energy_threshold and not(in_maturation_period?) then
        add_to_tabu_list(genome)
        offspring ← generate_offspring()
        update_population(offspring)
        genome ← replace_genome(offspring)
        controller ← assign_as_controller(genome)
        energy ← default_initial_energy
    end if
end loop
```

3.3 Experimental Methodology

In this section, we define our experimental methodology, and we describe the three tasks used in the study: aggregation, phototaxis, and integrated navigation and obstacle avoidance. Our experiments serve as a means to determine if and how odNEAT evolves controllers for solving the specified tasks, the complexity of solutions evolved, and the efficiency of odNEAT when compared with rtNEAT and IM-$(\mu + 1)$.

3.3.1 Experimental Setup

We use JBotEvolver (Duarte et al., 2014) to conduct our simulation-based experiments. JBotEvolver is an open-source, multirobot simulation platform, and neuroevolution framework. The simulator is written in Java and implements 2D differential drive kinematics.

In our experimental setup, the simulated robots are modelled after the e-puck (Mondada et al., 2009), a 7.5 cm in diameter differential drive robot capable of moving at speeds of up to 13 cm/s. Each robot is equipped with infrared sensors that multiplex obstacle sensing and communication between robots at a range of up to 25 cm.[1] The sensors of the robots are used in the three tasks to detect walls and other robots. In real e-pucks, the wall sensors can be implemented with active infrared sensors, while the robot sensors can be implemented using the e-puck range & bearing board (Gutiérrez et al., 2008). In the phototaxis task, robots are also given the ability to detect the light source, which can be provided by the fly-vision turret or by the omnidirectional vision turret.[2] Each sensor and each actuator are subject to noise, which is simulated by adding a random Gaussian component within $\pm 5\%$ of the sensor saturation value or actuation value. Each robot also has an internal sensor that enables it to perceive its current virtual energy level.

Each robot is controlled by an ANN produced by the evolutionary algorithm being tested. The controllers are discrete-time recurrent neural networks with connection weights in the interval $[-10, 10]$. The inputs of the ANN are the normalised readings in the interval $[0, 1]$ from the sensors mentioned above. The output layer has two neurons, whose values are linearly scaled from $[0, 1]$ to $[-1, 1]$. The scaled output values are used to set the signed speed of each wheel (positive in one direction, negative in the other). The activation function is the logistic function. The three algorithms start with simple networks with no hidden nodes, and with each input neuron connected to every output neuron. Evolution is therefore responsible for adding new neurons and new connections, both feed-forward and recurrent.

A group of five robots operates in a square arena surrounded by walls. The size of the arena was chosen to be 3 x 3 meters. Every 100 ms of simulated time, each robot executes a control cycle. In Table 3.2, we summarise the characteristics common to the three tasks. Regarding the configuration of the evolutionary algorithms, the parameters are the same for odNEAT, rtNEAT, and IM-$(\mu + 1)$: crossover rate - 0.25, mutation rate - 0.1, add neuron rate - 0.03, add connection rate - 0.05, and weight mutation magnitude - 0.5. The parameters were found experimentally: new connections have to be added more frequently than new neurons, and evolution tends to perform better if neural networks are recombined and augmented in a parsimonious manner. odNEAT is configured with a maturation period of 500 control cycles (50 seconds). Preliminary experiments showed this parameter value to provide an appropriate trade-off between the reliability of fitness estimates and the speed of convergence towards good solutions. A solution previously found to be poor is removed from the tabu list if a similar genome is not among the last 50 genomes received by the robot.

3.3.2 Preliminary Performance Tuning

We conducted a series of preliminary tests across all three tasks as a means to analyse the dynamics of odNEAT, rtNEAT, and IM-$(\mu + 1)$. Firstly, we tested both odNEAT and rtNEAT with: (i) a dynamic compatibility

[1] The original e-puck infrared range is 2-3 cm (Mondada et al., 2009). In real e-pucks, the *liblrcom* library, available at `http://www.e-puck.org`, extends the range up to 25 cm and multiplexes infrared communication with proximity sensing.

[2] See the Extensions section at `http://www.e-puck.org/`.

Table 3.2: Summary of the experimental details

Group size	5 robots
Broadcast range	25 cm
Control cycle frequency	100 ms
Arena size	3 x 3 meters
Simulation length	100 hours
Runs per configuration	30
odNEAT population size	40 genomes per internal population
IM-($\mu + 1$) population size	40 genomes per internal population
rtNEAT population size	200 genomes
Neural network weight range	[-10, 10]
Inputs range	[0,1]
Outputs range	[0,1], rescaled to [-1, 1]
Activation function	Logistic function

threshold and a target number of species from 1 to 10, and (ii) a fixed compatibility threshold $\delta = 3.0$ for speciating the population, with corresponding coefficients set as in previous studies using NEAT (Stanley and Miikkulainen, 2002; Stanley, 2004). Results were found to be similar, and we therefore decided to use a fixed compatibility threshold.

Secondly, we verified that the periodic production of new controllers in rtNEAT and in IM-($\mu + 1$) led to incongruous group behaviour. Regular substitution of controllers caused the algorithms to perform poorly in collective tasks that explicitly required continuous group coordination and cooperation, such as the aggregation task. In IM-($\mu + 1$), ruptures of collective behaviour were caused by having every robot changing to a new controller at the same time. In rtNEAT, rupture of collective behaviour was still significant but less accentuated, as only one controller was substituted at regular time intervals. We compared the average fitness score of consecutive groups of controllers executed on the robots. In the majority of the substitutions, the performance of new controllers was worse than the performance of the previous ones. In these cases, differences in the fitness scores were statistically significant in the aggregation task for rtNEAT and for IM-($\mu + 1$) ($\rho < 0.01$, Mann-Whitney test), and in the phototaxis task for IM-($\mu + 1$) ($\rho < 0.05$).

To provide a fair and meaningful comparison of performance, we modified the condition for creating new offspring. rtNEAT and IM-($\mu + 1$) were modified to produce offspring when a robot's energy level reaches the minimum threshold (zero in our experiments), that is, when a new controller is necessary. In the IM-($\mu + 1$) algorithm, we also observed collisions in the assignment of innovation numbers. To reduce the chance of collisions, we replaced the random assignment of innovation numbers by high-resolution timestamps, as in odNEAT. We verified that the modified versions of rtNEAT and IM-($\mu + 1$) consistently outperformed their original, non-modified counterparts by evolving final solutions with higher fitness scores ($\rho < 0.05$, Mann-Whitney, in the aggregation and phototaxis tasks), and that were found in fewer evaluations on average.

3.3.3 Aggregation

In an aggregation task, dispersed agents must move close to one another so that they form a single cluster. Aggregation plays an important role in a number of biological systems (Camazine et al., 2001). For instance, several social animals use aggregation to increase their chances of survival, or as a precursor of other collective behaviours. We study aggregation because it combines different aspects of multirobot tasks, namely distributed

search, coordinated movement, and cooperation. Aggregation is also related to a number of real-world robotics tasks. For instance, self-assembly and collective transport of heavy objects require aggregation at the site of interest (Groß and Dorigo, 2009).

In the initial configuration, robots are placed in random positions at a minimum distance of 1.5 meters between neighbours. Robots are evaluated based on a set of criteria that include the presence of robots nearby, and the ability to explore the arena and move fast. The initial virtual energy level of each controller E is set to 1000 units and limited to the range $[0, 2000]$ units. At each control cycle, E is updated according to the following equation:

$$\frac{\Delta E}{\Delta t} = \alpha(t) + \gamma(t) \tag{3.2}$$

where t is the current control cycle, and $\alpha(t)$ is a reward proportional to the number n of different genomes received in the last $P = 10$ control cycles. In our experiments, we use $\alpha(t) = 3 \cdot n$ energy units. As robots executing odNEAT exchange candidate solutions to the task, the number of different genomes received is used to estimate the number of robots nearby. The second component of the equation, $\gamma(t)$, is a factor related to the quality of movement computed as:

$$\gamma(t) = \begin{cases} \text{-1} & \text{if } v_l(t) \cdot v_r(t) < 0 \\ \Omega_s(t) \cdot \omega_s(t) & \text{otherwise} \end{cases} \tag{3.3}$$

where $v_l(t)$ and $v_r(t)$ are the left and right wheel speeds, $\Omega_s(t)$ is the ratio between the average and maximum speed, and $\omega_s(t) = \sqrt{v_l(t) \cdot v_r(t)}$ rewards controllers that move fast and straight at each control cycle. Robot and controller details are summarised in Table 3.3. To determine when robots have aggregated, we sample the number of robot clusters at regular intervals throughout the simulation as in Bahgeçi and Sahin (2005). Two robots are part of the same cluster if the distance between them is less than or equal to their sensor range (25 cm).

Table 3.3: Aggregation controller details

Input neurons: 18	
	8 for IR robot detection (range: 25 cm)
	8 for IR wall detection (range: 25 cm)
	1 for energy level reading
	1 for reading the number of different genomes received in the last $P = 10$ control cycles
Output neurons: 2	Left and right motor speeds

3.3.4 Integrated Navigation and Obstacle Avoidance

Navigation and obstacle avoidance is a classic task in evolutionary robotics. Robots have to simultaneously move as straight as possible, maximise wheel speed, and avoid obstacles. The task is typically studied using only one robot. In multirobot experiments, each robot poses as an additional, moving obstacle for the other robots. In a confined environment, such as our enclosed arena, this task implies an integrated set of actions,

and consequently a trade-off between avoiding obstacles in sensor range and maintaining speed and forward movement. Navigation and obstacle avoidance is therefore essential for autonomous robots operating in real-world environments, and it provides the basis to more sophisticated behaviours such as path planning or traffic rules (Cao et al., 1997).

Initially, robots are placed in random locations and with random orientations, drawn from a uniform distribution. The virtual energy level E is limited to the range $\in [0, 100]$ units. When the energy level reaches zero, a new controller is generated and assigned the default energy value of 50 units. During task execution, E is updated every 100 ms according to the following equation, adapted from Floreano and Mondada (1994):

$$\frac{\Delta E}{\Delta t} = f_{norm}(V \cdot (1 - \sqrt{\Delta v}) \cdot (1 - d_r) \cdot (1 - d_w)) \tag{3.4}$$

where V is the sum of rotation speeds of the two wheels, with $0 \leq V \leq 1$. $\Delta v \in [0, 1]$ is the normalised absolute value of the algebraic difference between the signed speed values of the wheels. d_r and d_w are the highest activation values of the infrared sensors for robot detection and for wall detection, respectively. d_r and d_w are normalised to a value between 0 (no robot/wall in sight) and 1 (collision with a robot or wall).

The four components encourage respectively, motion, straight line displacement, robot avoidance, and wall avoidance. The function f_{norm} maps linearly from the domain $[0, 1]$ into $[-1, 1]$. Robot and controller details are summarised in Table 3.4.

Table 3.4: Navigation and obstacle avoidance controller details

Input neurons: 17	
	8 for IR robot detection (range: 25 cm)
	8 for IR wall detection (range: 25 cm)
	1 for energy level reading
Output neurons: 2	Left and right motor speeds

3.3.5 Phototaxis

In a phototaxis task, robots have to search and move towards a light source. We use a dynamic version of the phototaxis task. Every five minutes of simulated time, the light source is moved instantaneously to a new random location. In this way, robots have to continuously search for and reach the light source, which eliminates behaviours that find the light source by chance. As in the navigation task, robots start at random locations and with random orientations drawn from a uniform distribution. The virtual energy level $E \in [0, 100]$ units, and controllers are assigned an initial value of 50 units. At each control cycle, the robot's virtual energy level E is updated as follows:

$$\frac{\Delta E}{\Delta t} = \begin{cases} S_r & \text{if } S_r > 0.5 \\ 0 & \text{if } 0 < S_r \leq 0.5 \\ penalty & \text{if } S_r = 0 \end{cases} \tag{3.5}$$

where $penalty$ = -0.01, and S_r is the maximum value of the readings from light sensors, between 0 (no light) and 1 (brightest light). Light sensors have a range of 50 cm and robots are therefore only rewarded if they are

close to the light source. Remaining IR sensors detect nearby walls and robots within a range of 25 cm. Details are summarised in Table 3.5.

Table 3.5: Phototaxis controller details

Input neurons: 25	
	8 for IR robot detection (range: 25 cm)
	8 for IR wall detection (range: 25 cm)
	8 for light source detection (range: 50 cm)
	1 for energy level reading
Output neurons: 2	Left and right motor speeds

3.3.6 Treatments

We compare the performance of odNEAT with the performance of rtNEAT and of IM-($\mu + 1$) in the three tasks. We analyse: (i) the number of evaluations, that is, the number of controllers tested by each robot before one that solves the task is found, (ii) the task performance in terms of fitness score, (iii) the complexity of solutions evolved, and (iv) their generalisation capabilities. The number of evaluations is measured because it is independent of the potentially different evaluation times used by the algorithms and thus ensures a meaningful comparison. To the best of our knowledge, there is no general metric for ANN complexity. We use the effective number of parameters in each network, C_{fp}, which is defined as the sum of the number of connections and the number of neurons that have a bias value. This measure of complexity is used in a number of heuristics for the back-propagation algorithm to determine, for instance, a suitable size for the training set (Haykin, 1999).

We use the two-tailed Mann-Whitney test to compute statistical significance of differences between sets of results because it is a non-parametric test, and therefore no strong assumptions need to be made about the underlying distributions. Success rates are compared using the two-tailed Fisher's exact test, a non-parametric test suitable for this purpose (Fisher, 1925).

For each of the three tasks considered and each algorithm evaluated, we conduct 30 independent evolutionary runs. When multiple comparisons are performed using the results obtained in each set of 30 runs (number of evaluations, fitness scores, complexity of solutions evolved, and their generalisation capabilities), we adjust the ρ-value using the two-stage Hommel method (Hommel, 1988), a more powerful and less conservative modification of the classic Bonferroni correction method (Blakesley, 2008).

3.4 Experimental Results

In this section, we present and discuss the experimental results. The section is divided into two subsections. In the first subsection, we compare the performance of odNEAT with the performance of rtNEAT. In the second subsection, we compare the performance of odNEAT with the performance of IM-($\mu + 1$).

3.4.1 Comparing odNEAT and rtNEAT

We start by comparing the performance of odNEAT and rtNEAT to evaluate the quality of our algorithm with respect to a centralised state-of-the-art neuroevolution method. Table 3.6 summarises the number of evaluations

and fitness scores of controllers that solve the task for both odNEAT and rtNEAT. In terms of evaluations, results show comparable performance even though there is a slight advantage in favour of rtNEAT. On average, odNEAT requires each robot to evaluate approximately 6 to 8 controllers more than rtNEAT. Differences in the number of evaluations are not statistically significant ($\rho \geq 0.05$, Mann-Whitney).

With respect to the fitness scores, both odNEAT and rtNEAT typically evolve high performing controllers as summarised in Table 3.6.[3] In two of the three tasks assessed, odNEAT tends to outperform rtNEAT. In the aggregation task and in the navigation task, odNEAT produces superiorly performing controllers ($\rho < 0.05$, Mann-Whitney). In the phototaxis task, rtNEAT evolves superior controllers ($\rho < 0.05$, Mann-Whitney).

Table 3.6: Comparison of the number of evaluations and of the fitness scores of solutions to the task (out of 100) between odNEAT and rtNEAT. Values listed are the avg. ± std. dev. over 30 independent runs for each experimental configuration

Task	Method	Number of evaluations	Fitness score
Aggregation	odNEAT	103.7 ± 80.9	89.2 ± 4.8
	rtNEAT	95.5 ± 60.4	83.4 ± 11.9
Navigation	odNEAT	23.6 ± 19.2	93.0 ± 9.2
	rtNEAT	17.6 ± 11.1	89.9 ± 11.4
Phototaxis	odNEAT	40.9 ± 24.1	85.7 ± 6.4
	rtNEAT	34.8 ± 16.3	89.6 ± 6.3

odNEAT differs from rtNEAT in the sense that it has a distributed and decentralised nature, and therefore does not assess the group-level information from the global perspective. To unveil the evolutionary dynamics of the two algorithms, we first analyse the stepping stones that lead to the final solutions, that is, how the evolutionary search proceeds through the high-dimensional genotypic search space for the respective algorithms. A number of methods have been proposed to perform such analysis (Kim and Moon, 2003; Vassilev et al., 2000), and they are mainly based on visualisations of the structure of fitness landscapes or the fitness score of the best individuals over time. To visualise the intermediate genomes produced by odNEAT and rtNEAT, and how the algorithms traverse the search space *with respect to each other*, we use *Sammon's nonlinear mapping* (Sammon Jr., 1969). Contrarily to visualisation and dimensionality reduction techniques such as Principal Components Analysis (Kittler and Young, 1973), and Self-organising Maps (Kohonen, 1982), Sammon's mapping aims to preserve the original distances between elements in the mapping to a lower dimensional space.

Sammon's mapping performs a point mapping of high-dimensional data to two- or three-dimensional spaces, such that the structure of the data is approximately preserved. The algorithm minimises the error measure E_m, which represents the disparity between the high-dimensional distances δ_{ij} and the resulting distance d_{ij} in the lower dimension for all pairs of points i and j. E_m can be minimised by a steepest descent procedure to search for a minimum error value, and it is computed as follows:

$$E_m = \frac{1}{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\delta_{ij}} \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{(\delta_{ij} - d_{ij})^2}{\delta_{ij}} \tag{3.6}$$

[3]The fitness scores for the aggregation task shown in Table 3.6 are linearly mapped from $[0, 2000]$ to $[0, 100]$, which is the range of the fitness score for the other two tasks.

Using Sammon's mapping, we project the genomes representing intermediate controllers tested over the course of evolution by odNEAT and rtNEAT. In order to obtain a clearer visualisation, we do not map *all* genomes produced during the course of evolution. Instead, we only map genomes that have a genomic distance $\delta > 4.5$ when compared with already recorded genomes. This criterion was found experimentally to ensure that the two-dimensional space has a representative selection of genomes while being readable. The output of the mapping are x and y coordinates for every genome. The distance in the high-dimensional space δ_{ij} between two genomes i and j is based on genomic distance as used by odNEAT and rtNEAT for speciation. On the other hand, the distance between two points in the two-dimensional visualisation space is computed as their Euclidean distance.

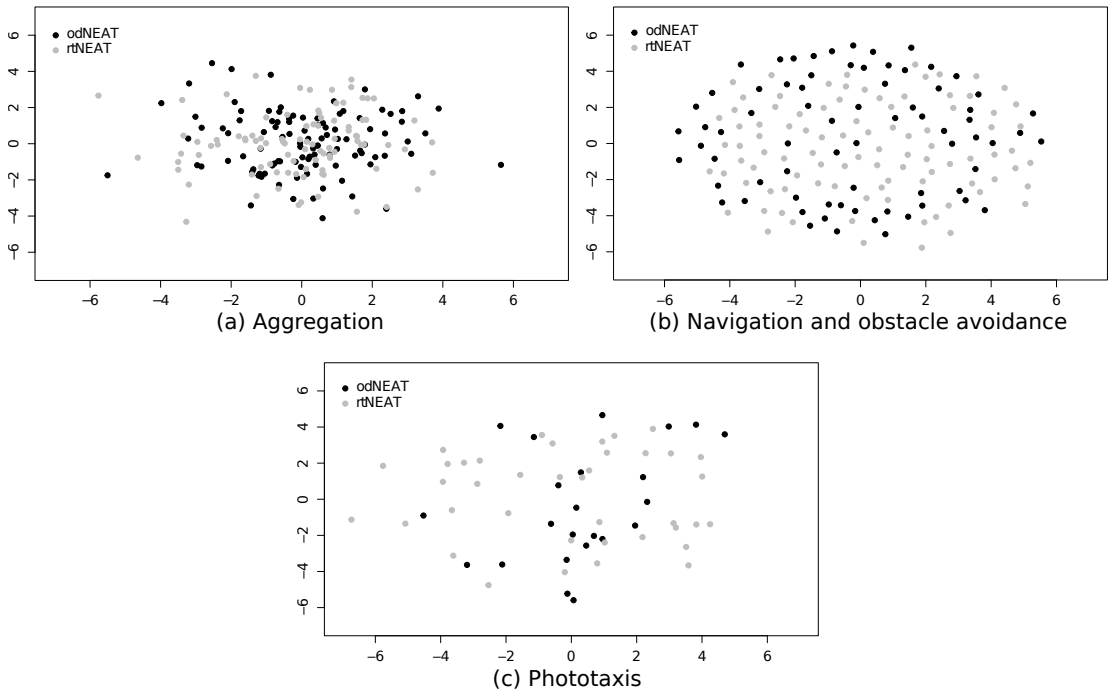


Figure 3.1: Sammon's mapping. (a) Aggregation task. Sammon's mapping of 103 genotypes evolved by odNEAT (black) and 95 genotypes evolved by rtNEAT (gray). (b) Navigation and obstacle avoidance task. Sammon's mapping of 72 genotypes evolved by odNEAT (black) and 91 genotypes evolved by rtNEAT (gray). (c) Phototaxis task. Sammon's mapping of 21 genotypes evolved by odNEAT (black) and 39 genotypes evolved by rtNEAT (gray).

In Figure 3.1, we show the Sammon's mapping for the three tasks. The error values are $E_m = 0.086$ for the aggregation task, $E_m = 0.075$ for the navigation and obstacle avoidance task, and $E_m = 0.082$ for the phototaxis task. The low error values indicate that the distances between genomes are well-preserved by the mapping. In the aggregation task, Sammon's mapping shows a similar exploration of the search space. odNEAT and rtNEAT explore identical regions in the two-dimensional space, and evolve a comparable number of genomes matching those regions: 103 evolved by odNEAT vs. 95 evolved by rtNEAT. On the other hand, in the navigation task, odNEAT evolves fewer genomes matching the analysed regions of the search space: 72 vs. 91 evolved by rtNEAT. That is, rtNEAT covers more regions of the genotypic search space. A similar trend is also observed in the phototaxis task, in which rtNEAT evolves 39 genomes vs. 21 evolved by odNEAT.

Since odNEAT relies exclusively on local information, the evolutionary algorithm executing on each robot tends to do a more confined exploration of the search space. In the aggregation task, robots are in close proximity and they therefore continuously exchange genomes. In such case, the distributed and decentralised dynamics of odNEAT resemble, to some extent, the dynamics of a centralised algorithm. Hence, the exploration of the search space performed by odNEAT and rtNEAT is similar, quantitatively and qualitatively. In the other two tasks, because robots tend to be further apart, there is typically more pressure to evolve solutions by using the information available in the population of each individual robot.

Neural Complexity and Generalisation Performance

Table 3.7 lists the complexity reached by each evolutionary method in the final successful solutions. odNEAT's evolutionary dynamics also lead to the synthesis of simpler solutions than in rtNEAT, both in terms of neurons and connections added through evolution. Differences in the neural complexity of evolved solutions are significant in all experimental configurations ($\rho < 0.01$, Mann-Whitney).

Table 3.7: Summary of the neural complexity added through evolution from the initial topology by odNEAT and rtNEAT. Connections added and neurons added refer to the avg. ± std. dev. over 30 independent runs for each experimental configuration

Task	Method	Connections added	Neurons added	Avg. C_{fp}
Aggregation	odNEAT	6.3 ± 5.6	5.5 ± 5.0	11.8
	rtNEAT	11.3 ± 11.1	9.3 ± 8.9	20.6
Navigation	odNEAT	6.6 ± 0.9	3.1 ± 0.3	9.7
	rtNEAT	10.0 ± 1.4	4.9 ± 0.7	14.9
Phototaxis	odNEAT	2.3 ± 0.7	1.1 ± 0.3	3.4
	rtNEAT	5.4 ± 1.9	2.5 ± 0.9	7.9

In artificial neural network training, it has been shown that, among a set of solutions for a given task, less complex networks tend to have better generalisation performance (Schmidhuber, 1997). To compare the generalisation capabilities of odNEAT and rtNEAT, we restart each task 100 times per original evolutionary run. In the task restarts, each robot maintains its controller and further evolution is not allowed. Task restarts are generalisation tests that enable us to assess if robots can continuously operate after several redeployments. The generalisation tests involve both the flexibility to solve the task starting from different initial conditions, and the ability to operate in conditions potentially not experienced during the evolutionary phase. A group of robots passes the generalisation test if it continues to solve the task, that is, if the virtual energy level of any of the robots in the group does not reach zero (see Sections 3.3.3, 3.3.4, and 3.3.5). Each generalisation test has a maximum duration of 100 hours of simulated time.

Table 3.8 lists the generalisation performance of odNEAT and rtNEAT. In general, odNEAT presents an interesting capacity to generalise and execute in different conditions. odNEAT outperforms rtNEAT by approximately 13.5 percentage points in the aggregation task and by 4.0 percentage points in the navigation task, as it successfully solves 406 and 120 tests more, respectively. Differences in successful generalisation tests are statistically significant in the two tasks ($\rho < 0.0001$ in the aggregation task, and $\rho < 0.01$ in the navigation task, Fisher's exact test). In the phototaxis task, rtNEAT yields better generalisation performance and successfully

Table 3.8: Generalisation performance of controllers evolved by odNEAT and rtNEAT in the three tasks. The generalisation performance refers to the avg. ± std. dev. of the success rate for each set of 100 task restarts

Task	Method	Generalisation performance (%)	Successful tests
Aggregation	odNEAT	75.1 ± 32.6	2253/3000
	rtNEAT	61.6 ± 32.4	1847/3000
Navigation	odNEAT	73.4 ± 23.3	2202/3000
	rtNEAT	69.4 ± 31.9	2082/3000
Phototaxis	odNEAT	60.2 ± 33.5	1806/3000
	rtNEAT	62.3 ± 19.0	1870/3000

solves 64 tests more than odNEAT, which corresponds to approximately 2.1 percentage points. Differences are considerably smaller than in the other two tasks, and are not statistically significant ($\rho > 0.05$).

Overall, the analysis performed in this section shows that odNEAT yields performance levels comparable to those of rtNEAT in terms of the number of evaluations necessary to evolve solutions, and of the task performance of the final controllers. In addition, odNEAT consistently evolves controllers with relatively low complexity and superior generalisation capabilities that can potentially adapt and operate in different deployment scenarios without further evolution.

3.4.2 Comparing odNEAT and IM-$(\mu + 1)$

In this section, we compare the performance of odNEAT and IM-$(\mu + 1)$. Experiments conducted with the IM-$(\mu + 1)$ algorithm serve as a means to compare odNEAT with an algorithm with a similar fundamental characteristic: the decentralised online evolution of neural topologies and weights.

Comparison of performance is shown in Table 3.9. In the aggregation task, odNEAT and IM-$(\mu + 1)$ evolve solutions to the task at similar rates. Differences in the number of evaluations between the two algorithms are not statistically significant ($\rho \geq 0.05$, Mann-Whitney). In the remaining two tasks, the navigation task and the phototaxis task, odNEAT significantly outperforms IM-$(\mu + 1)$ with respect to the number of evaluations ($\rho < 0.001$, Mann-Whitney). odNEAT requires approximately 54% of the evaluations needed by IM-$(\mu + 1)$ in the navigation task, and 45% of the evaluations in the phototaxis task. Furthermore, odNEAT always evolves controllers that yield significantly higher fitness scores ($\rho < 1 \cdot 10^{-4}$, Mann-Whitney).

Table 3.9: Comparison of the number of evaluations and of the fitness scores of solutions to the task (out of 100) between odNEAT and IM-$(\mu + 1)$. Values listed are the avg. ± std. dev. over 30 independent runs for each experimental configuration

Task	Method	Number of evaluations	Fitness score
Aggregation	odNEAT	103.7 ± 80.9	89.2 ± 4.8
	IM-$(\mu + 1)$	100.8 ± 21.9	75.8 ± 10.5
Navigation	odNEAT	23.6 ± 19.2	93.0 ± 9.2
	IM-$(\mu + 1)$	43.6 ± 10.8	89.5 ± 0.4
Phototaxis	odNEAT	40.9 ± 24.1	85.7 ± 6.4
	IM-$(\mu + 1)$	91.0 ± 30.6	77.6 ± 9.9

Table 3.10 lists the neural complexity of evolved solutions. Results show that ANNs evolved by odNEAT are also less complex than those evolved by IM-($\mu + 1$). Both algorithms evolve networks with recurrent and feed-forward connections. Differences in C_{fp} values are statistically significant across the three tasks ($\rho < 0.001$, Mann-Whitney). Overall, the IM-($\mu + 1$) algorithm is biased towards large networks. Since there is no fitness cost in adding new neurons and connections, IM-($\mu + 1$) consistently generates large neural topologies. The growth is due to the structural mutation operators as: (i) *each* connection gene has a fixed equal probability of generating a new connection gene in the same genome, and (ii) insertion of new neuron genes is based on the duplication and differentiation of a neuron gene *and* its incoming and outgoing connection genes. This form of growth leads to networks that consistently have more connections than neurons added through evolution, as listed in Table 3.10. Since larger networks have more parameters and need more time to be optimised, either by the adjustment of weighting parameters or the removal of unnecessary neurons and connections through mutation, the algorithm tends to require more evaluations to find solutions than odNEAT.

Table 3.10: Summary of the neural complexity added through evolution from the initial topology by odNEAT and IM-($\mu + 1$). Connections added and neurons added refer to the avg. $\pm$ std. dev. over 30 independent runs for each experimental configuration. In the IM-($\mu + 1$) algorithm, the size proportionate addition of new connections and the duplication of neurons, and of their incoming and outgoing connections, lead to networks that consistently have a large number of connections (see text for details)

Task	Method	Connections added	Neurons added	Avg. C_{fp}
Aggregation	odNEAT	6.3 $\pm$ 5.6	5.5 $\pm$ 5.0	11.8
	IM-($\mu + 1$)	23.6 $\pm$ 12.2	2.2 $\pm$ 0.4	25.8
Navigation	odNEAT	6.6 $\pm$ 0.9	3.1 $\pm$ 0.3	9.7
	IM-($\mu + 1$)	36.7 $\pm$ 14.4	2.5 $\pm$ 0.6	39.2
Phototaxis	odNEAT	2.3 $\pm$ 0.7	1.1 $\pm$ 0.3	3.4
	IM-($\mu + 1$)	26.0 $\pm$ 14.9	1.8 $\pm$ 0.5	27.8

Importantly, the niching scheme in odNEAT protects topological innovations and also prevents bloating of genomes: species with smaller genomes are maintained in the population as long as their fitness is competitive, and smaller networks are thus not replaced by larger ones unnecessarily. The successive generation of new candidate controllers leads to a progressive optimisation of existing structure in each robot's internal population with parsimonious addition of structure. In effect, the structure of each intermediate solution represents a search space of parameter values that evolution must optimise. The more complex the structure, the higher the number of parameters that must be optimised simultaneously. If the structural complexity can be minimised, the dimensionality of the search spaces explored along the path to a solution is reduced, and evolution can more efficiently optimise the intermediate solutions. Such an approach will generally lead to performance gains in terms of: (i) speed of convergence towards the final solution, that is, the number of evaluations, and (ii) the task performance of the intermediate and final solutions evolved. This hypothesis is supported by the results listed in Table 3.9 and in Table 3.10, which show that odNEAT evolves less complex networks that always outperform those evolved by IM-($\mu + 1$) in terms of their ability to solve the task, even when the evaluations necessary to evolve a solution are comparable (as in the aggregation task).

In the generalisation experiments, we observe that odNEAT evolves controllers that also display a generalisation performance superior to the controllers evolved by IM-($\mu + 1$), as listed in Table 3.11. Depending

Table 3.11: Generalisation performance of controllers evolved by odNEAT and IM-($\mu + 1$) in the three tasks. The generalisation performance refers to the avg. $\pm$ std. dev. of the success rate for each set of 100 task restarts

Task	Method	Generalisation performance (%)	Successful tests
Aggregation	odNEAT	75.1 $\pm$ 32.6	2253/3000
	IM-($\mu + 1$)	46.1 $\pm$ 24.5	1382/3000
Navigation	odNEAT	73.4 $\pm$ 23.3	2202/3000
	IM-($\mu + 1$)	55.2 $\pm$ 36.2	1657/3000
Phototaxis	odNEAT	60.2 $\pm$ 33.5	1806/3000
	IM-($\mu + 1$)	46.0 $\pm$ 14.3	1379/3000

on the task, odNEAT outperforms IM-($\mu + 1$) between approximately 14 percentage points and 29 percentage points, as robots executing odNEAT successfully solve 427 to 871 generalisation tests more. Differences in the number of successful generalisation tests are statistically significant ($\rho < 0.0001$ in the three tasks, Fisher's exact test). The results of the generalisation tests are coherent with those obtained in Section 3.4.1, which showed that odNEAT evolves neural networks with comparatively high generalisation capabilities. During long-term operation in the field, robots may experience environmental conditions not seen during the evolutionary phase. Therefore, producing robust controllers that can adapt to new circumstances without further evolution is advantageous, and has been subject to increasing interest (Lehman et al., 2013a). Another important aspect in robotic systems is the robustness to failures (Christensen et al., 2009). The following section is devoted to understanding the properties of odNEAT with respect to the algorithm's ability to adapt to faults in the sensors, and the impact of each algorithmic component on performance.

3.5 Assessing odNEAT: Fault Injection Experiments and Ablation Studies

In the previous section, we experimentally compared the performance of odNEAT with the performance of rtNEAT and IM-($\mu + 1$). In this section, we further assess odNEAT's robustness and features. We focus on two aspects: (i) odNEAT's ability to address long-term self-adaptation when there are faults in the robots' sensors, and (ii) the impact of each algorithmic component on performance.

3.5.1 Adaptation Performance

The experimental protocol for the fault injection experiments described in this section is defined in coherence with the results of Carlson et al. (2004), which analysed the reliability of 15 mobile robots in terms of physical failures. For small robots operating in the field, such as the models considered in this study, there is an overall frequency of 0.10 failures per hour, and 12% of failures affect the sensors. In their study, the authors do not distinguish between complete failure and partial failure. In our experiments, we assume sensor failures as damaging the sensor completely, and feeding a zero signal into the neural network during subsequent readings.

In the fault injection experiments, we conduct 30 independent runs using a group of 5 robots. We double the duration of each run to 200 hours of simulated time to examine the long-term effects of injected faults. Every hour of simulated time, faults are injected with probability 0.10, in which case one randomly chosen

operational physical sensor becomes faulty with probability 0.12. Each robot starts out with the controller evolved in the experiments described in the previous section, but further evolution *is* allowed. The goal of using evolved controllers is to separate learning to solve the task from learning to overcome sensor faults.

To assess the effects of faults in the sensors, we analyse: (i) the number of controllers produced to cope with a given percentage of faulty sensors, and (ii) the operation time (age) of the controllers used by the robots during the experiments. The number of controllers produced is an indicator of the difficulty of the evolutionary process to adapt the behaviour of robots when faults are present. Complementarily, the operation time of controllers relates to the number of faults they can tolerate, thereby indicating the robustness of solutions evolved.

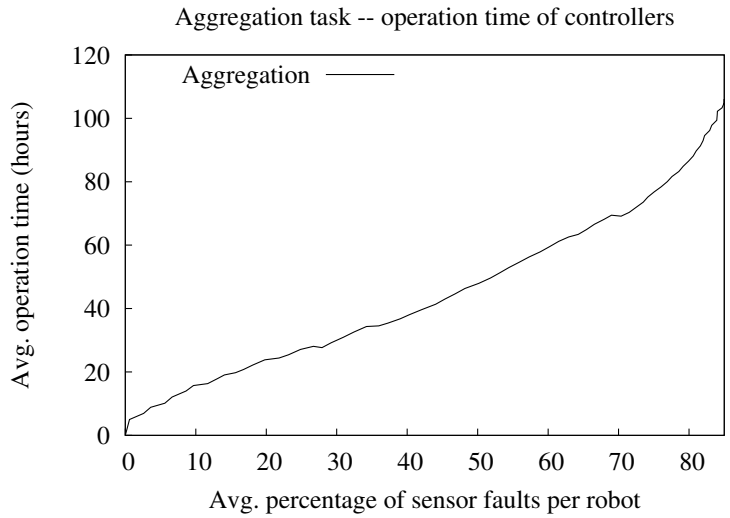


Figure 3.2: Fault injections during the aggregation task: average operation time (age) of controllers executing at a given time, and subject to a given percentage of faults in the sensors. The operation time increases linearly, with a gentle slope, thereby indicating that new controllers are rarely necessary.

The controllers more robust to faults are those evolved in the aggregation task. On average, robots can sustain faults in approximately 85% of physical sensors, which corresponds to 20 out of 24 physical sensors. After this point, the experiments are terminated because the 200 hours limit is reached. Figure 3.2 shows the operation time of controllers during the experiments. As more faults are injected, the operation time continues to increase linearly, with a gentle slope, which indicates that new controllers are rarely necessary. In effect, the final controllers operate for more than 100 consecutive hours.

The high robustness to faults in the aggregation task is due to the robot's behaviour and to the task requirements. Robots form a single group and there is, therefore, considerable sensory information available. As long as robots can sense other robots with one or two sensors, faults in the remaining sensors have virtually no effect on performance. The high degree of tolerance to faults is due to the exchange of genomes between robots. As described in Section 3.3.3, the number of genomes received by a given robot is used as an estimation of the number of robots nearby, and is part of the virtual energy level and fitness score computations. Because only the physical sensors of the robots are affected by faults, the *virtual* energy level sensor and the "received genomes" sensor function normally and robots continue to solve the task.

Figure 3.3 shows the number of controllers produced and the operation time of the controllers in both

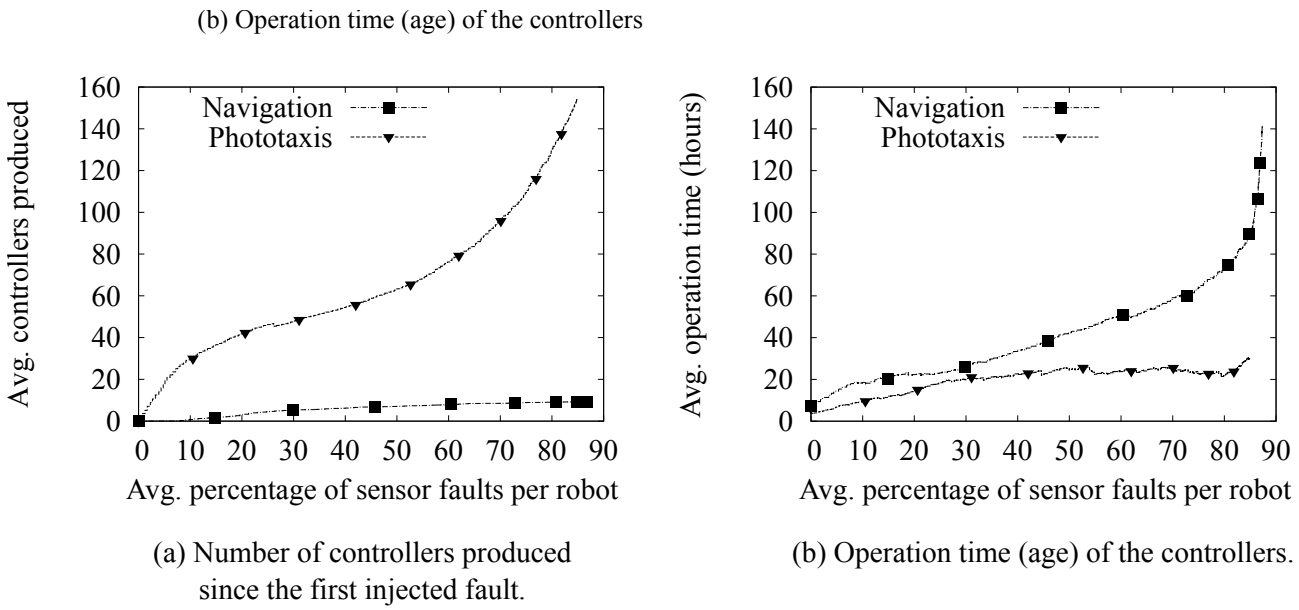


Figure 3.3: Summary of the results concerning the injection of faults in the robots' sensors. (a) Number of controllers produced since the first fault was injected. (b) Operation time (age) of each controller executing at a given time.

the navigation and phototaxis tasks. In these two tasks, odNEAT can also adapt to cope with failures in approximately 80% to 85% of the sensors, corresponding to a maximum of 14 sensors in the navigation and obstacle avoidance task, and 20 sensors in the phototaxis task. However, contrary to the aggregation task, robots have to evolve new controllers more often to handle the new sensory conditions. In the phototaxis task, with the increasing number of faults, odNEAT progressively tests more controllers as a means to synthesise solutions for the task. As shown in Figure 3.3(a), when the percentage of faults reaches 50%, each robot had evolved on average 60 new controllers. For a higher percentage of faults, it becomes increasingly more difficult for odNEAT to evolve a suitable solution. When faults affect 75% and 85% of the sensors, the number of controllers evaluated grows to approximately 100 and 160, respectively. This result indicates that robots experience significant difficulties when more than 50% of the sensors are not functional. However, as supported by the approximately stable average operation time of the group for a percentage of faults greater than 50%, shown in Figure 3.3(b), new controllers are able to sustain a moderate degree of faults in sensors before failing. In terms of neural augmentation, odNEAT continuously adds new structure in response to the sensor faults. odNEAT adjusts and augments neural topologies from an average of 3.4 parameters added through evolution, for solutions not subject to faults, to approximately 28.2 parameters for solutions subject to faults in 85% of the sensors.

In the integrated navigation and obstacle avoidance task, robots are robust to faults. As shown in Figure 3.3(a), the initial controller only becomes unable to solve the task when approximately 10% of the sensors fail. From the first fault until 85% of the sensors fail, each robot tested 12.7 new controllers on average. Thus, even though a significant portion of the sensors are faulty, robots only evaluated relatively few new controllers during the *entire* fault injection experiments. In the synthesis of new controllers, odNEAT does not augment neural topologies substantially as the complexity of solutions is relatively stable. Neural complexity of solutions

is augmented from 9.7 parameters on average added through evolution, for solutions not subject to faults, to 10.8 parameters for solutions with 85% faults in the sensors. Nonetheless, it should be noted that there is a duality in the task. With the progressive injection of faults, robots display a better navigation performance but a worse obstacle avoidance performance. This is due to the obstacle avoidance fitness component relying on sensor readings, for determining if there are obstacles nearby, while the navigation component relies on the speed of the wheels, see Equation (3.4), which are not affected by faults. An important consideration is that when the percentage of faults is greater than 67%, the less effective obstacle avoidance performance results in occasional collisions of robots with other robots and with the walls of the arena if the direction of contact is not sensed.

Overall, the analysis in this section shows that odNEAT is able to evolve adaptive solutions that can cope with faults in the robots' sensors. Distinct task requirements lead to different responses in the adaptation process. Robots executing odNEAT: (i) are almost unaffected by faults in the aggregation task, (ii) progressively change their behaviour in the navigation and obstacle avoidance task, and (iii) experience more difficulties in the phototaxis task, but are still able to adapt to the new sensory conditions.

3.5.2 Ablation Studies

In order to determine the impact of each algorithmic component in odNEAT, we perform a series of ablation studies. We use aggregation as the task for ablation studies. Aggregation was found to be the hardest task, as shown in Table 3.6 and Table 3.9, and thus provides a degree of difficulty suitable for comparing the ablated versions of odNEAT with the complete version of odNEAT. We conduct experiments in six distinct experimental setups: (i) with a minimal internal population of size 2, (ii) without the exchange of genomes between robots, (iii) without the tabu list, (iv) without the maturation period, (v) without the niching scheme, and (vi) without the crossover operator. Note that the number of genomes received by one robot is directly used in the fitness calculation in the aggregation task, see Section 3.3.3. To implement the ablation study in which robots do not exchange genomes, we allow robots to communicate but the genomes received are not included in the evolutionary process, that is, they are discarded immediately after the energy level and fitness score calculations have been performed.

Table 3.12: odNEAT ablations summary. The table lists the simulation time (in hours), the number of evaluations, and the success rate of each experimental configuration. Results listed for each configuration are the avg. ± std. dev. over 30 independent runs

Method	Sim. Time	Evaluations	Success Rate
Minimal population	16.4 ± 21.2	236.2 ± 213.8	22/30
No exchange of solutions	11.3 ± 12.0	156.0 ± 103.8	24/30
No tabu list	8.9 ± 12.2	133.6 ± 119.2	28/30
No maturation	13.9 ± 22.5	211.3 ± 287.8	29/30
No niching	41.1 ± 17.6	240.2 ± 103.0	25/30
No crossover	7.8 ± 6.5	127.8 ± 91.6	28/30
Full odNEAT	6.2 ± 5.6	103.7 ± 80.9	30/30

Results are shown in Table 3.12, and averaged over 30 independent runs for each configuration. Results in

this table exclude runs that failed to find sustainable behaviours within 100 hours of simulated time. Simulation time is measured to complement the number of evaluations. In odNEAT, controllers execute as long as they are able to solve the task, that is, the virtual energy level is above the minimum threshold, and the duration of evaluations therefore tends to vary.

The most critical algorithmic component of odNEAT is the internal population. Differences in performance between full odNEAT and odNEAT with a minimal population are statistically significant in terms of the number of evaluations ($\rho < 0.01$, Mann-Whitney), and success rate ($\rho = 4.6 \cdot 10^{-3}$, Fisher's exact test). The size of the population is important because the population maintains a local view of the system's history and provides the basis for evolution. With a minimal population, evolution is limited to a small set of genomes, in this case 2. This experimental setup causes a significant instability in the evolutionary process as evolution is much slower and may even be incapable of exploring enough of the solution space to find successful solutions, hence the comparatively low success rate, and high simulation time and number of evaluations.

The exchange of genomes between robots is also a crucial feature in odNEAT's performance. Differences are statistically significant with respect to the number of evaluations ($\rho < 0.05$, Mann-Whitney) and the success rate ($\rho = 2.37 \cdot 10^{-2}$, Fisher's exact test). To quantify to what extent is a robot dependent on the genomes it receives from other robots, we analyse the origin of the information stored in each population when executing the full, non-ablated odNEAT version. When evolution is ended, 73.4% of the genomes maintained in each internal population originated from other robots, whereas 26.6% of the genomes stored were evolved by the robots themselves. The final solutions executed by each robot to solve the task have on average 77.0% of matching genes. Moreover, 17.6% of these solutions have more than 90% of their genes in common. The average weight difference between matching connection genes is of 4.8, with each weight $w \in [-10, 10]$. Local exchange of genetic information is therefore a crucial part in the odNEAT's evolutionary dynamics that serves as a substrate for speeding up the evolutionary process and for collective problem solving.

Without the tabu list, odNEAT achieves a success rate of 28/30 runs. Differences in the number of evaluations and in the success rate are not significant. However, the comparatively high standard deviation in the simulation time and in the number of evaluations reflects the instability of the evolutionary process when the tabu list is not used. The tabu list keeps the evolutionary process from cycling around in an unfruitful neighbourhood of the solution space, which may happen given that odNEAT evolves controllers only based on local information. In effect, by rejecting genomes similar to those that failed, the tabu list promotes topological diversity in the population and positively smooths evolution, thereby providing a relevant contribution to the performance of odNEAT.

The maturation period defines a lower bound of activity in the environment, giving the new offspring a chance to spread their genome. Ablating the maturation period provides results significantly different in terms of the number of evaluations ($\rho < 0.05$, Mann-Whitney), but not significant with respect to the success rate. Without the maturation period, good solutions are potentially lost forever and evolution is decelerated. Robots are still be capable of solving the task in most experiments, but exhibit a clear deficit in performance.

By ablating the niching scheme of odNEAT, performance is considerably affected. Robots solve the task in 25/30 runs. Differences in the success rate are not significant. When odNEAT is able to evolve a solution to the task, it requires substantially more time, and robots evaluate significantly more controllers ($\rho < 0.001$, Mann-

Whitney). Without the crossover operator, odNEAT fails to find solutions for the task in 2 of the 30 runs. When odNEAT is able to find a solution without crossover, each robot is subject to approximately 24 additional evaluations, equivalent to 18.9%. Differences in the number of evaluations are statistically significant ($\rho < 0.01$, Mann-Whitney). The series of ablation studies involving genetic operators show that both the crossover and the niching scheme have a significant effect on performance, and further support a number of conclusions in the literature: (i) the crossover operator is an advantage for neuroevolution algorithms when performed appropriately, for instance by resorting to innovation numbers (Stanley and Miikkulainen, 2002), and (ii) genotypic diversity mechanisms such as speciation and fitness sharing can improve performance in evolutionary robotics tasks (Mouret and Doncieux, 2012).

The ablation of any component of odNEAT's components leads to a less efficient algorithm. Arguably, the most important conclusion drawn from the ablation studies is that all components of odNEAT contribute to the algorithm's performance as an efficient online, distributed, and decentralised neuroevolution algorithm.

3.6 Summary

In this chapter, we have presented and extensively assessed an algorithm called odNEAT for evolutionary online learning and adaptation in groups of robots. odNEAT implements the online evolutionary process according to a physically distributed island model. Each robot optimises an internal population of genomes, and there is the exchange of genetic information between robots. An important advantage of odNEAT over the majority of existing online neuroevolution algorithms is that the optimisation of both neural parameters and topology is under evolutionary control and an appropriate network topology is the result of a continuous evolutionary process.

We compared odNEAT with rtNEAT and with IM-(μ+1) in three tasks: (i) aggregation, (ii) integrated navigation and obstacle avoidance, and (iii) phototaxis. Our study produced three main results. Firstly, odNEAT yields performance levels similar to rtNEAT, a state-of-the-art neuroevolution algorithm, and outperforms the IM-(μ + 1) algorithm. Secondly, compared with rtNEAT and IM-(μ + 1), odNEAT exhibits a higher evolutionary pressure towards neural controllers with low complexity and with superior generalisation capabilities. Thirdly, our experiments showed that individual robots executing odNEAT can successfully adapt to cope with faults in the sensors. Depending on the task requirements, robots can tolerate different degrees of faults and, if necessary, evolve new controllers and progressively modify their behaviour in a completely autonomous manner. Overall, our study showed that odNEAT is an efficient and robust algorithm, and a promising approach for online evolution in multirobot systems.

Chapter 4

Dynamics of Online Evolution of Robotic Controllers

Over the past decade, different approaches to online evolution in multirobot systems have been introduced (Silva et al., 2016d). Notwithstanding, different aspects in the dynamics of online evolution have been largely left unstudied (Silva et al., 2013, 2015b), including:

- **The properties at the level of individual neurons:** online evolution studies have been almost exclusively based on artificial neural networks composed of a variation of the neuronal model introduced by McCulloch and Pitts (1943), that is, summing neurons. With advances in biology, multiplicative-like operations have been found in neurons as a means to process non-linear interactions between sensory inputs (Koch, 2004; Cazenille et al., 2012). In machine learning, multiplicative neurons have shown to increase the computational power and storage capacities of neural networks (Durbin and Rumelhart, 1989; Schmitt, 2002).

- **The scalability properties of online evolution with respect to group size:** when the online evolutionary algorithm is distributed across a group of robots, one common assumption is that online evolution inherently scales with the number of robots (Watson et al., 2002). Generally, the idea is that the more robots are available, the more evaluations can be performed in parallel, and the faster the evolutionary process (Watson et al., 2002). The dynamics of the online evolutionary algorithm itself, and common issues that arise in evolutionary algorithms from population sizing such as convergence rates and diversity (de Jong, 2006) have, however, not been considered. Furthermore, besides ad-hoc experiments with large groups of robots, see Bredeche et al. (2012) for examples, there has been no systematic study on the scalability properties of online evolutionary algorithms across different tasks.

In this chapter, we report simulation-based experiments designed to investigate the dynamics of online evolution of controllers at two different scales. At the microscopic scale, we study the dynamics of distinct neuronal models. At the macroscopic scale, we investigate the scalability properties of online evolution with respect to group size. The chapter is organised as follows. In Section 4.1, we investigate and compare the performance and robustness of neural network-based controllers using summing neurons, multiplicative neurons, and a combination of the two. In Section 4.2, we present a case study on the scalability of online evolution in

four tasks with varying numbers of robots. In Section 4.3, we summarise our research contributions and provide concluding remarks.

4.1 Neuronal Model

In discrete-time artificial neural networks, the classic summing unit is the most commonly used neuronal model. The summing neuronal model performs the computation as follows:

$$a_i = f\Big(\sum_{j=1}^{N} w_{ji} \cdot x_j + w_0\Big) \,. \tag{4.1}$$

where a_i is the activation level of a given neuron i, and f is the activation function applied on the weighted sum of inputs from incoming neurons x_j, plus the bias value w_0. The activation function f can take the form of, for instance, a threshold function or a sigmoid function. However, relying exclusively on sums of weighted inputs potentially limits performance and learning capabilities when complex and/or non-linear interactions are considered (Schmitt, 2002; Koch, 2004), as in the case of robotics.

In machine learning, multiplicative neurons were introduced more than 25 years ago (Durbin and Rumelhart, 1989). Examples include the pi-sigma and the sigma-pi units (Schmitt, 2002), and the more general product unit neuronal model (Durbin and Rumelhart, 1989), which we study. In the product unit model, the activation of a neuron i is computed as follows:

$$a_i = f\Big(\prod_{j=1}^{N} x_j^{w_{ji}}\Big) \,. \tag{4.2}$$

with notations similar to Equation 4.1. The number of exponents N gives the *order* of the neuron, thus denoting the artificial neural network as a *higher-order neural network*. The exponents are real values, in which negative exponents enable division operations.

The potential benefits of the product unit model have been widely discussed in the literature. Using gradient descent methods, Durbin and Rumelhart (1989) concluded that product units have superior information and learning capabilities compared to summing units. Complementarily, Schmitt (2002) analysed the gains in information processing and learning capabilities of product unit neurons in terms of solution complexity and computational power. Despite the positive results, only recently the potential benefits of product units were investigated in an evolutionary robotics context. Cazenille et al. (2012) performed the offline evolution of neural network-based controllers for the *coupled inverted pendulum* (Hamann et al., 2011), a benchmark for modular robotics scenarios. The authors investigated the interplay between microscopic properties such as the neuronal model, and macroscopic properties such as modularity and repeating motifs in neural networks. Surprisingly, their results suggested that product units may be counter-productive when used alone. If neural networks display regularities such as modularity, then product units may lead to better fitness scores, while requiring fewer evaluations for evolving a solution.

4.1.1 Experimental Methodology

In this section, we define our experimental methodology to investigate the dynamics and potential benefits of multiplicative neuronal models, and of summing neuronal models, separately and combined, in online evolution

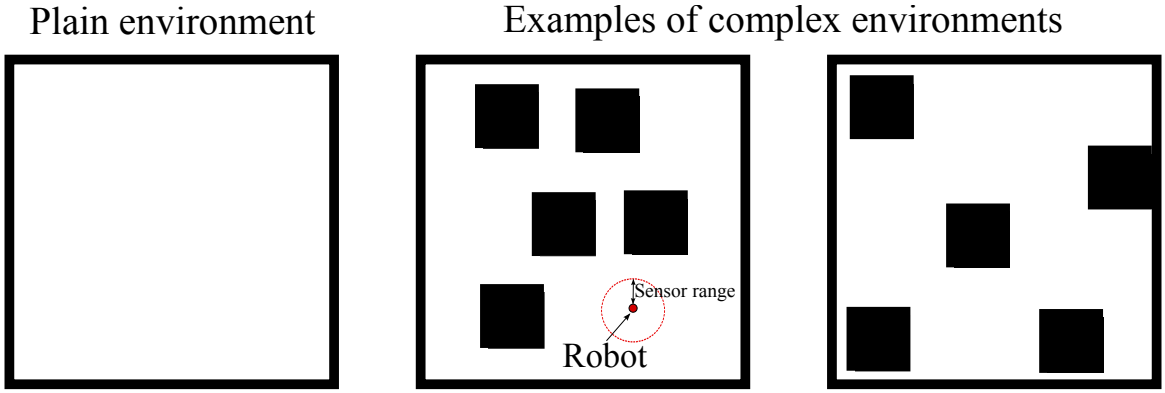


Figure 4.1: The two types of environments, plain and complex, used to evolve controllers. Each of the arenas measures 3 x 3 meters. The dark areas denote physical obstacles, while the white areas denote the arena surface on which robots can navigate.

in multirobot systems. Specifically, we define the experimental setup, the treatments, and how we characterise the behaviours evolved during the experiments.

Experimental Setup

We conduct a simulated experiment in which a group of robots modelled after the e-puck (Mondada et al., 2009) must perform an integrated navigation and obstacle avoidance task, as described in Section 3.3, in environments of distinct complexity. The goal of the experiments is to assess how different neuronal models fare in a simple task and environment, and to analyse if and how performance differs if the task is carried out in a more challenging environment.

The first environment is a plain arena, in which the only obstacles are the robots and the walls that confine the arena. The second environment is an arena with five additional obstacles. Each obstacle has dimensions of 0.5 x 0.5 meters. The additional obstacles are of the same material as walls, and intuitively increase the difficulty of the task by reducing the area for navigation. In each evolutionary run conducted in the second environment, obstacles are placed at random locations. Examples of environments are shown in Figure 4.1.

We perform three sets of evolutionary experiments in each environment. In one set of experiments, neurons added through structural mutation are multiplicative product units. In the second set of experiments, neurons introduced are summing units. In the third set of experiments, each new neuron has an equal probability of 0.5, sampled from a uniform distribution, of being either a multiplicative or summing neuron. Multiplicative product units may therefore be combined with summing units, and introduce the ability for computing weighted sums of products, and vice-versa. Other experimental parameters are set as described in Section 3.3.

Treatments

For each experimental configuration, we perform 30 independent evolutionary runs. Each run lasts 100 hours of simulated time. We compare the performance of the three controller models based on three aspects: (i) the number of evaluations, that is, the number of controllers tested by each robot before a solution to the task is found, (ii) the complexity of solutions evolved, and (iii) their generalisation capabilities. Similarly to the treatments described in Section 3.3, we use the two-tailed Mann-Whitney test to compute statistical signifi-

cance of differences between sets of results, and success rates are compared using the two-tailed Fisher's exact test (Fisher, 1925). Additionally, statistical dependence between two variables is computed using the non-parametric Spearman's rank correlation coefficient. When multiple comparisons are performed, we adjust the ρ-value using the two-stage Hommel method (Hommel, 1988).

Characterisation of Behavioural Diversity

Ultimately, online evolution of controllers synthesises the *behavioural* control of robots. To characterise and compare behaviours evolved, we use a generic Hamming distance-based behavioural metric based on the mapping between sensors and actuators. This measure has shown to be, at least, as efficient as domain-dependent behavioural metrics (Mouret and Doncieux, 2012). The behaviour metric is based on the set of sensor readings and actuation values ϑ normalised into [0,1], as follows:

$$\vartheta = \Big[\{\mathbf{s}(t), \mathbf{a}(t)\}, t \in \{1, T\}\Big] \,. \tag{4.3}$$

where $\mathbf{s}(t)$ and $\mathbf{a}(t)$ are the sensor readings and actuation values at time t, respectively, and T is the number of observations. The binary version ϑ_{bin} of ϑ is computed as follows:

$$\vartheta_{bin} = \Big[\vartheta_{bin}(t), t \in \{1, T\}\Big] = \Big[\{\mathbf{s}_{bin}(t), \mathbf{a}_{bin}(t)\}, t \in \{1, T\}\Big] \,. \tag{4.4}$$

where each $s_{bin,i}(t) \in \mathbf{s}_{bin}(t)$ is defined as 1 if $s_i(t) > 0.5$ and 0 otherwise, and each $a_{bin,j}(t) \in \mathbf{a}_{bin}(t)$ is defined as 1 if $a_j(t) > 0.5$ and 0 otherwise. Indexes i and j respectively vary from 1 to the length of sequences $\mathbf{s}_{bin}(t)$ and $\mathbf{a}_{bin}(t)$. The Hamming distance between two behaviours is then computed as follows:

$$\sigma(\vartheta_1, \vartheta_2) = \sum_{t=1}^{T} h(\vartheta_{1,bin}(t), \vartheta_{2,bin}(t)) \,. \tag{4.5}$$

$$h(\vartheta_1, \vartheta_2) = \sum_{i=1}^{len(\vartheta_1)} 1 - \delta(\vartheta_1[i], \vartheta_2[i]) \,. \tag{4.6}$$

where $len(\vartheta_1) = len(\vartheta_2)$ denotes the length of the binary sequences ϑ_1 and ϑ_2, and $\delta(i, j)$ is the Kronecker delta defined as $\delta(i, j) = 1$ if i = j, and 0 otherwise.

We extend the generic Hamming distance between sensor readings and actuation values to capture the *intra-behaviour* distance as follows:

$$\sigma(\vartheta_{bin}) = \sum_{t=1}^{T} h(\vartheta_{bin}(t-1), \vartheta_{bin}(t)) \,. \tag{4.7}$$

$\sigma(\vartheta_{bin})$ captures the differences between consecutive observations of the relation between sensor readings and actuation values, thereby approximating to what extent the behaviour of a robot varies during task execution.

4.1.2 Experimental Results

In this section, we present and discuss the experimental results. We compare the performance of the three controller models, their generalisation capabilities, and the evolutionary dynamics when different neuronal models are considered.

Table 4.1: Number of evaluations for the three types of controllers considered in the two environments (average ± std. dev.).

Controller	Plain environment	Complex environment
Summing	23.63 ± 19.16	22.61 ± 18.68
Multiplicative	26.08 ± 21.10	32.02 ± 24.93
Hybrid	21.54 ± 20.53	27.81 ± 29.16

Comparison of Performance

The number of evaluations for the three types of controllers is listed in Table 4.1. Differences are not significant for any comparison ($\rho > 0.05$, Mann-Whitney). In the plain environment, evolution of hybrid controllers requires fewer evaluations to synthesise solutions for the task. In the complex environment, solutions are synthesised faster when summing controllers are evolved. In both the plain and the complex environments, evolution of multiplicative controllers requires on average the largest number of evaluations to evolve solutions. Interestingly, both hybrid controllers and multiplicative controllers are affected by the increase in environment complexity. Summing controllers, on the other hand, require an approximate number of evaluations in the two environments.

By analysing the complexity of solutions evolved, we observed simple neural topologies for solving the task in each environment, as listed in Table 4.2. Overall, multiplicative controllers present the least complex topologies, in equality with summing controllers in the complex environment. Despite requiring less structure to solve the task, the number of evaluations to synthesise suitable multiplicative neurons is higher, as discussed previously and shown in Table 4.1. This is due to multiplicative neurons requiring a finer-grain adjustment of parameters. Compared to summing neurons, multiplicative controllers required more adjustments of connection weights through mutation and, therefore, a higher number of evaluations.

Complementarily, hybrid controllers present the most complex topologies. The crossover operator manipulates topological structure involving different neural dynamics, that is, summing and multiplicative neurons. We analysed the effects of evolutionary operators and found a more accentuated decrease in fitness scores when hybrid controllers are recombined. This effect is progressively eliminated as new neurons and new connections are added to the network. Nonetheless, despite differences in terms of neural complexity and number of evaluations, each experimental configuration lead to the evolution of high-scoring controllers. The average fitness score of solutions varies from 91.31 to 94.59 in the plain environment, and from 91.24 to 95.49 in the complex environment. Differences in fitness scores are not statistically significant for any comparison.

Table 4.2: Neural complexity of solutions evolved. Neurons and connections added through evolution (average ± std. dev.).

	Plain environment		Complex environment	
Controller	Neurons	Connections	Neurons	Connections
Summing	3.14 ± 0.35	6.59 ± 0.90	1.17 ± 0.41	2.57 ± 1.03
Multiplicative	2.16 ± 0.40	4.62 ± 0.94	1.21 ± 0.42	2.84 ± 1.09
Hybrid	3.16 ± 0.39	6.56 ± 1.13	3.11 ± 0.32	6.47 ± 0.79

Table 4.3: Generalisation performance of each controller model. The table lists the average generalisation capabilities of each group of five robots, and the total of generalisation tests solved successfully.

	Plain environment		Complex environment	
Controller	Generalisation (%)	Succ. tests	Generalisation (%)	Succ. tests
Summing	73.40 ± 23.33	2202/3000	45.30 ± 30.36	1359/3000
Multiplicative	85.87 ± 13.07	2576/3000	57.03 ± 26.09	1711/3000
Hybrid	84.10 ± 24.24	2523/3000	80.60 ± 14.77	2418/3000

Testing for generalisation

To analyse the generalisation capabilities of the different types of controllers, we conducted a series of generalisation tests. The generalisation tests are identical to those carried out in the experiments described in Section 3.4.1. For each evolutionary run conducted previously, we restarted the task 100 times using the controllers evolved and not allowing further evolution. Each task restart is a generalisation test serving to assess if robots can continuously operate after redeployment, and for evaluating the ability to operate in conditions not experienced during the evolution phase. A group of robots passes a generalisation test if it continues to solve the task, that is, if the virtual energy level of none of the robots reaches zero. Each generalisation test has the same duration as the evolutionary phase, 100 hours of simulated time. In the complex environment, the five obstacles are placed in random locations in each test.

In Table 4.3, we show the generalisation performance of each controller model. In the plain environment, multiplicative controllers outperform summing controllers by 12% as they solve 374 tests more. Differences in successful and unsuccessful generalisation tests are statistically significant ($\rho < 0.0001$, Fisher's exact test). The hybrid controllers display high generalisation performance, similar to multiplicative controllers. Differences between these two controllers are not statistically significant. In the complex environment, multiplicative controllers also outperform summing controllers by approximately 12% due to solving 352 tests more ($\rho < 0.0001$, Fisher's exact test). Importantly, hybrid controllers significantly outperform both summing controllers and multiplicative controllers ($\rho < 0.0001$, Fisher's exact test), and maintain approximate generalisation levels across the two environments. In this way, the higher number of evaluations to synthesise hybrid controllers in the complex environment is compensated for by the increase in generalisation performance caused by the interplay between summing neurons and multiplicative neurons in the same neural architecture.

The generalisation performance of hybrid controllers is a factor particularly important in the case of online evolution. Results obtained indicate that hybrid controllers can adapt more efficiently than multiplicative and summing controllers alone to contexts not explicitly encountered during evolution, thus avoiding the continuation of the evolutionary process. In this way, hybrid controllers revealed to be advantageous as they are high-scoring controllers with superior generalisation capabilities that require a competitive number of evaluations.

Analysis of Genotypic and Behavioural Search Space

To unveil the evolutionary dynamics of summing neurons and of multiplicative neurons, we compared how the evolutionary search proceeds through the high-dimensional genotypic search space. To visualise the intermediate genotypes produced when using the two neuronal models, and how they traverse the search space *with respect to each other*, we use *Sammon's nonlinear mapping* (Sammon Jr., 1969) (see Section 3.4.1 for details). The

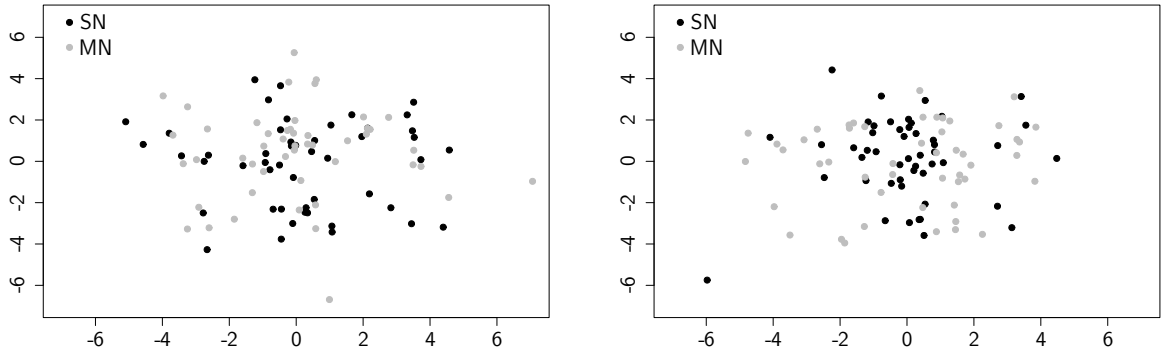


Figure 4.2: Combined Sammon's mapping of intermediate genotypes. Left: Plain environment. Right: Complex environment. Genotypes representing networks using summing neurons are marked in black, and genotypes representing networks with multiplicative neurons are marked in gray.

distance in the high-dimensional space δ_{ij} between two genotypes i and j is based on genomic distance as used in odNEAT. The distance between two points in the two-dimensional space is their Euclidean distance.

In Figure 4.2, we show the Sammon's mapping of genotypes. In order to obtain a clearer visualisation, and a consistent and representative selection of genotypes, we map the 50 most genetically different genotypes, which in turn represent the 50 most diverse topologies with respect to each experimental configuration. In general, there is a balanced exploration of the search space, as different controller models discover similar regions of the genotypic search space. Genotypes representing hybrid controllers also explore similar regions of the search space. The progress throughout the evolutionary process is therefore similar in the three controller models, indicating that the fitness landscape is not significantly altered by the use of different neuronal models. In this way, the dynamics of neuronal models may lead to the evolution of different *behaviours*, which in turn would account for differences in performance. To verify this hypothesis, we analysed the behaviour space explored by solutions evolved. We used the two generic Hamming distance-based behaviour metrics described in Section 4.1.1. For each solution, we analysed the behaviour during 10,000 seconds of simulated time with a sampling rate of 10 seconds, resulting in a behaviour vector of length 1000.

Table 4.4 summarises the inter-behaviour distances and intra-behaviour distances between final solutions to the task. In the plain environment, robots using hybrid controllers yield significantly higher inter-behaviour distances, and thus display behaviours that are more diverse than those of robots using summing controllers ($\rho < 0.01$, Mann-Whitney) and multiplicative controllers ($\rho < 0.0001$, Mann-Whitney). In turn, summing controllers also produce more diverse behaviours than multiplicative controllers ($\rho < 0.0001$, Mann-Whitney). With the increase of environmental complexity, all types of controllers display more diverse behaviours than in the plain environment. Summing controllers yield inter-behaviour distances significantly higher than those of robots using multiplicative controllers or hybrid controllers ($\rho < 0.0001$, Mann-Whitney). Differences between hybrid controllers and multiplicative controllers are not statistically significant. In all experimental configurations, there is a strong monotonic correlation between the novelty of the behaviour in terms of average inter-behaviour distance and the degree of intra-behaviour variance ($\rho > 0.98$, Spearman's correlation). There is, however, no clear correlation between the inter- and intra-behaviour distances and fitness scores ($-0.09 < \rho < 0.34$, Spearman's correlation), as both types of behaviours are related with high-performing controllers.

Table 4.4: Summary of inter-behaviour distances and intra-behaviour distances between final controllers to the task.

Inter-behaviour distances	Plain environment		Complex environment	
Controller	Mean	Max.	Mean	Max.
Summing	51.7	446.0	104.2	726.0
Multiplicative	40.7	413.0	83.1	813.0
Hybrid	63.5	448.0	85.3	511.0
Intra-behaviour distances	Plain environment		Complex environment	
Controller	Mean	Max.	Mean	Max.
Summing	51.8	446.0	104.0	798.0
Multiplicative	40.6	418.0	81.9	886.0
Hybrid	63.8	478.0	85.5	554.0

4.2 Scalability in Groups of Robots

In this section, we study the scalability properties of online evolution of robotic controllers. We assess the dynamics and performance of odNEAT in four tasks involving groups of up to 25 simulated e-puck-like robots (Mondada et al., 2009): (i) an aggregation task, (ii) a dynamic phototaxis task, and (iii, iv) two foraging tasks with differing complexity.

4.2.1 Experimental Methodology

In this section, we define our experimental methodology, including the setup, the four tasks used in the study, treatments, and configuration of the scalability experiments in terms of the number of robots.

Experimental Setup

The robot sensor and actuator configurations for the tasks are listed in Table 4.5. For each task, the controller inputs are the readings from the sensors, normalised to the interval [0,1]. The output layer has two neurons whose values are linearly scaled from [0,1] to [-1,1] to set the signed speed of each wheel. In the two foraging tasks, a third output neuron sets the state of a gripper. The gripper is activated if the output value of the neuron is higher than 0.5, otherwise it is deactivated. If the gripper is activated, the robot collects the closest resource within a range of 2 cm, if there is any. Depending on the foraging task (see below), the robot may need to actively select which type of resources to collect and to avoid other types. The two foraging tasks are described in the following section. The aggregation task and the phototaxis task are implemented as described in Section 3.3.3 and in Section 3.3.5, respectively.

Foraging tasks

In a foraging task, robots have to search for and pick up objects scattered in the environment. Foraging is a canonical testbed in cooperative robotics domains, and is evocative of tasks such as toxic waste clean-up, harvesting, and search and rescue (Cao et al., 1997).

We setup a foraging task with different types of resources that have to be collected. Robots spend virtual energy at a constant rate and must learn to find and collect resources. When a resource is collected by a

Table 4.5: Controller details. Light sensors have a range of 50 cm (phototaxis task). Other sensors have a range of 25 cm.

Aggregation task – controller details	
Input neurons: 18	
	8 for IR robot detection
	8 for IR wall detection
	1 for energy level reading
	1 for reading the number of different genomes received
Output neurons: 2	Left and right motor speeds
Phototaxis task – controller details	
Input neurons: 25	
	8 for IR robot detection
	8 for IR wall detection
	8 for light source detection
	1 for energy level reading
Output neurons: 2	Left and right motor speeds
Foraging tasks – controller details	
Input neurons: 25	
	4 for IR robot detection
	4 for IR wall detection
	1 for energy level reading
	8 for resource A detection
	8 for resource B detection
Output neurons: 3	
	2 for left and right motor speeds
	1 for controlling the gripper

robot, a new resource of the same type is placed randomly in the environment so as to keep the number of resource constant throughout the experiments. We experiment with two variants of a foraging task: (i) one in which there are only type A resources, henceforth called *standard foraging task*, and (ii) one in which there are both type A and type B resources, henceforth called *concurrent foraging task*. In the concurrent foraging task, resources A and B have to be collected alternately. That is, besides learning the foraging aspects of the task, robots also have to learn to collect resources in the correct order. The energy level of each controller is initially set to 100 units, and limited to the range [0,1000]. At each control cycle, E is updated as follows:

$$\frac{\Delta E}{\Delta t} = \begin{cases} reward & \text{if right type of resource is collected} \\ penalty & \text{if wrong type of resource is collected} \\ \text{-0.02} & \text{if no resource is collected} \end{cases} \tag{4.8}$$

where $reward = 10$ and $penalty = -10$. The constant decrement of 0.02 means that each controller will execute for a period of 500 seconds if no resource is collected since it started operating. The *penalty* component applies only to the concurrent foraging task. To enable a meaningful comparison of performance when groups of different size are considered, the number of resources of each type is set to the number of robots multiplied by 10.

Configuration of Scalability Experiments and Treatments

We analyse the impact of the group size on the performance of odNEAT by conducting experiments with groups of 5, 10, 15, 20, and 25 robots. For each experimental configuration, we conduct 30 independent evolutionary

runs. Each run lasts 100 hours of simulated time. odNEAT parameters are set as in previous experiments. Robots operate in a square arena surrounded by walls. In the aggregation and phototaxis tasks, the area of the arena is increased proportionally to the number of robots (5 robots: 9 m^2, 10 robots: 18 m^2, ..., 25 robots: 45 m^2). Notice that if we maintained the same size of the environment, comparisons would not be meaningful given the variation of robot density. For instance, in the aggregation task, with the increasing density of robots in the environment, the task becomes easier to solve simply because robots encounter each other more frequently. In the phototaxis task, the number of light sources in the environment is also increased proportionally to the number of robots.

We use the two-tailed Mann-Whitney to compute differences between sets of results, and the two-stage Hommel method (Hommel, 1988) to adjust the ρ-value when multiple comparisons are performed.

4.2.2 Experimental Results

In this section, we present our experimental results. We analyse: (i) the task performance of controllers in terms of their individual fitness score, (ii) the number of evaluations, that is, the number of controllers tested by each robot before a solution to the task is found, and (iii) the corresponding evolution time.

Quality of the Solutions and Population-Mixing

We first compare the individual fitness scores of the final controllers (see Figure 4.3). In the aggregation task and in the phototaxis task, groups of 5 robots are typically outperformed by larger groups ($\rho < 0.01$ and $\rho < 0.0001$, respectively, Mann-Whitney). In the phototaxis task, groups of 25 robots also perform significantly better than groups with 20 robots ($\rho < 0.05$). Specifically, results suggest that a minimum of 10 robots is necessary for high-performing controllers to be evolved in a consistent manner. In the standard foraging task, the fitness score of the final controllers is comparable across the different experimental configurations. In the concurrent foraging task, increases in group size from 10 robots to 15 robots, and from 15 robots to 20 robots lead to significant improvements in performance ($\rho < 0.001$ and $\rho < 0.05$, respectively).

To quantify to what extent is a robot dependent on the candidate solutions it receives from other robots, we analyse the origin of the information stored in the population of each robot. In the phototaxis task, when capable solutions have been evolved approximately 86.85% (5 robots) to 93.95% (25 robots) of genomes maintained in each internal population originated from other robots, whereas the remaining genomes stored were produced by the robots themselves (analysis of the results obtained in the other tasks revealed a similar trend). The final solutions executed by each robot to solve the task have on average from 87.26% to 89.10% matching genes. Moreover, 39.73% (5 robots) to 47.70% (25 robots) of these solutions have more than 90% of their genes in common. Local exchange of candidate controllers therefore appears to be a crucial part in the evolutionary dynamics of decentralised online evolutionary algorithms because it serves as a substrate for collective problem solving. In the following section, we analyse how the exchange of such information enables online evolutionary algorithms to capitalise on increasingly larger groups of robots for faster evolution of solutions to the task.

Evaluations and Time Analysis

The distribution of evaluations per robot is shown in Figure 4.4. In the aggregation task, the number of evaluations required to evolve solutions to the task decreases as the group size is increased, and becomes

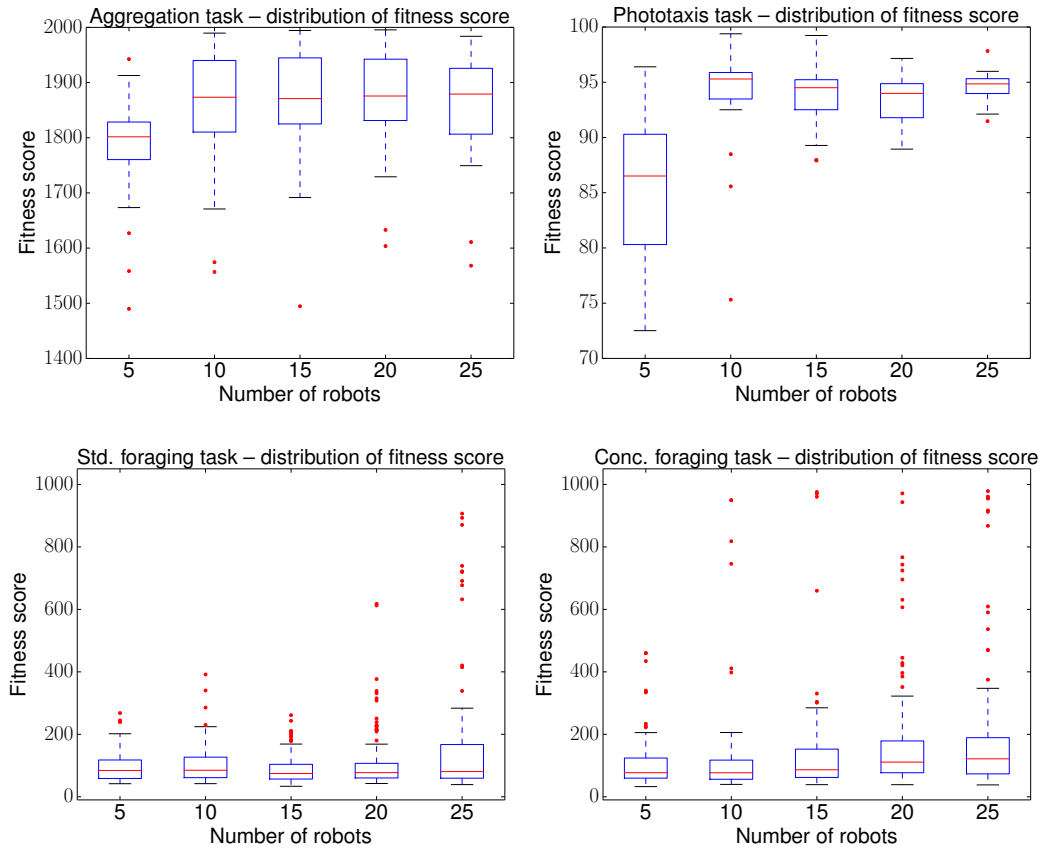


Figure 4.3: Distribution of the fitness score of the final controllers to the tasks. The lower limit and the upper limit of boxplots are respectively: 1671 and 1996 in the aggregation task; 73 and 99 in the phototaxis task; 34 and 284 in the standard foraging task; 33 and 347 in the concurrent foraging task.

significantly lower when the group size is increased from 10 to 15 robots. On average, the number of evaluations decreases from 104 for groups of 5 robots to 55 for groups of 25 robots. The mean evolution time is of 6.22 hours for groups of 5 robots, 2.34 hours for 10 robots, 1.80 hours for 15 robots, 1.48 hours for 20 robots, and 1.12 hours for 25 robots. Hence, adding more robots also enables a significant reduction of the evolution time ($\rho < 0.01$ for every group increment). With the increase in the size of the environment, there is a larger area to search for other robots and to explore. Task conditions become more challenging because, in relative terms, each robot senses a smaller portion of the environment. Robots are, however, still able to evolve successful controllers in fewer evaluations and less evolution time.

The speed up of evolution with the increase of group size also occurs in the phototaxis task. The number of evaluations is significantly reduced ($\rho < 0.0001$) with the increase of the group size from 5 to 10 robots (mean number of evaluations of 39 and 14, respectively). The mean evolution time is of 39.16 hours for groups of 5 robots, 9.51 hours for 10 robots, 7.20 hours for 15 robots, 6.30 hours for 20 robots, and 5.27 hours for 25 robots. Similarly to the number of evaluations, the evolution time yields on average a 4-fold-decrease when the group is enlarged from 5 to 10 robots ($\rho < 0.001$). Larger groups enable further improvements ($\rho < 0.001$ for increases up to 20 robots, $\rho < 0.01$ when group size is changed from 20 to 25 robots), but at comparatively smaller rates. Chiefly, the results of the aggregation task and of the phototaxis task show quantitatively distinct

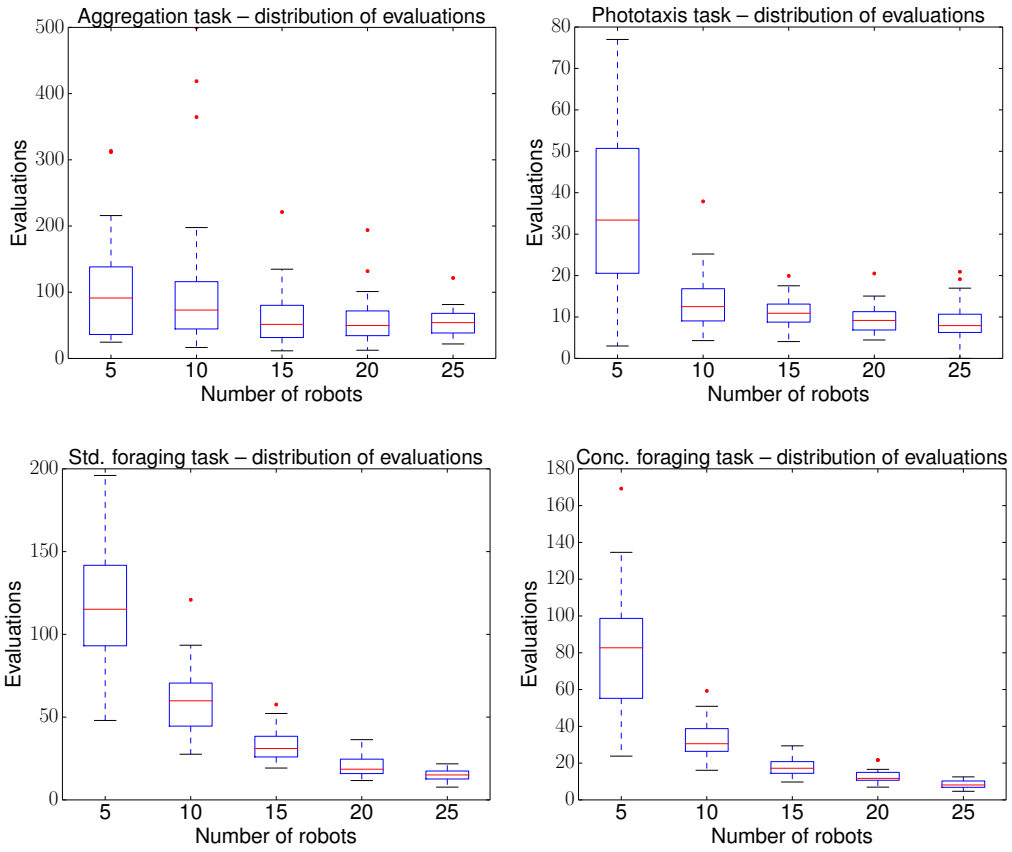


Figure 4.4: Distribution of evaluations in: (a) aggregation task, (b) phototaxis task, (c) standard foraging task, and (d) concurrent foraging task. The lower limit and the upper limit of boxplots are respectively: 12 and 216 in the aggregation task; 1 and 77 in the phototaxis task; 8 and 196 in the standard foraging task; 5 and 135 in the concurrent foraging task.

speed-ups of evolution when groups are enlarged.

With respect to the two foraging tasks, the distribution of the number of evaluations shown in Figure 4.4 is inversely proportional, with a gentle slope, to the number of robots in the group. For both tasks, differences in the number of evaluations are significant across all comparisons ($\rho < 0.001$). In effect, the number of evaluations is reduced on average: (i) from 115 evaluations (5 robots) to 15 evaluations (25 robots) in the standard foraging task, which corresponds to a 7.67-fold decrease in terms of evaluations, and (ii) from 82 evaluations (5 robots) to 8 evaluations in the concurrent foraging task, which amounts to a 10.25-fold decrease. These results show that decentralised online evolution can scale well in terms of evaluations, even when task complexity is increased.

Regarding the evolution time, results show a similar trend for both foraging tasks. On average, the evolution time varies from approximately 35 and 36 hours for groups of 5 robots to 21 and 23 hours for groups of 25 robots. That is, despite significant improvements in terms of the number of evaluations, the evolution time required to evolve the final controllers to the task is still prohibitively long. This result is due to the controller evaluation policy. Online evolution approaches typically employ a policy in which robots substitute controllers at regular time intervals, see Bredeche et al. (2012) for an example. This approach has been shown to lead to incongruous group behaviour and to poor performance in collective tasks that explicitly require continuous

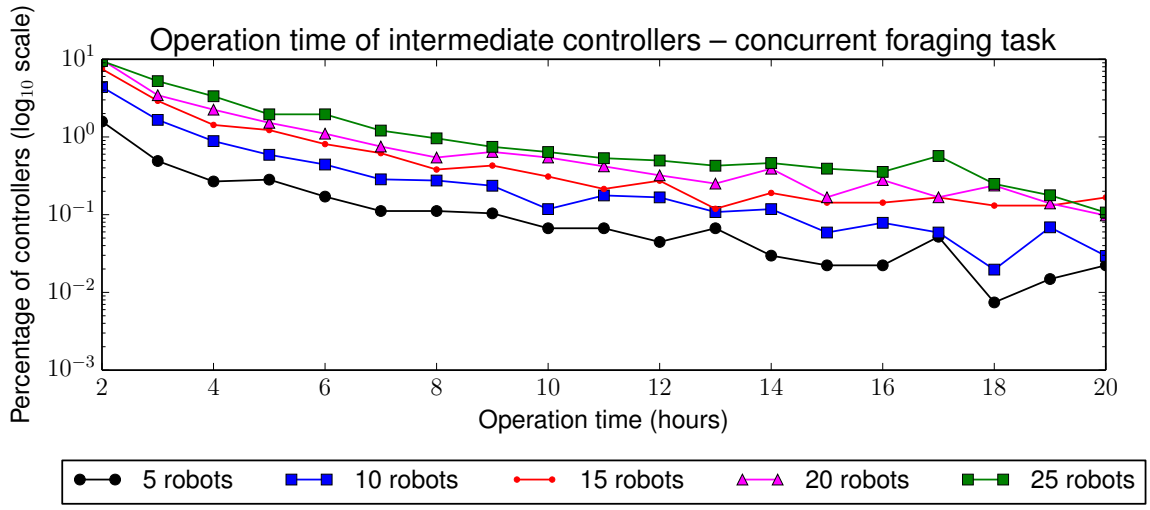


Figure 4.5: Operation time of intermediate controllers in the concurrent foraging task. 67% to 96% of intermediate controllers operate for few minutes before they fail (not shown for better plot readability).

collective coordination and cooperation (Silva et al., 2015e). odNEAT, on the other hand, adopts a different approach by allowing a controller to remain active as long as it is able to solve the task. A new controller is thus only synthesised if the current one fails. As shown in Figure 4.5 for the concurrent foraging task, the evaluation policy results in intermediate controllers that operate for a significant amount of time (the standard foraging task displays a similar trend). While 67% to 96% of intermediate controllers only operate for a few minutes (data not shown for better plot readability), there are a few intermediate controllers that operate up to 20 hours of consecutive time before they fail. Although such controllers yield high fitness scores comparable to those of the final solutions, typically 1% to 4% less, they delay the synthesis of more effective solutions.

4.3 Summary

In this chapter, we investigated the dynamics of online evolution of controllers in multirobot systems at two different scales. In the first case study, we compared the dynamics of multiplicative neurons, separately and when combined with summing neurons, in terms of number of evaluations, complexity of solutions evolved, task performance, generalisation capabilities, and diversity of behaviours evolved. The results of our experiments can be distilled into the following findings:

- Solutions are synthesised faster when summing neurons are used alone.
- Neural controllers using a combination of summing neurons and multiplications can provide competitive results in terms of number of evaluations, and exhibit significantly superior generalisation capabilities.

In the second case study, we focused on the dynamics at the macroscopic level and investigated the scalability properties and performance of online evolutionary algorithms. Our main results are the following:

- Larger groups of robots can enable: (i) superior task performance in terms of the fitness score, and (ii) significant improvements both in terms of the number of evaluations required to evolve solutions to the task and of the corresponding evolution time.

- While additional robots may further speed up evolution, there are specific group sizes after which speed-ups are comparatively smaller.
- There are specific conditions in which intermediate controllers are able to operate up to 20 hours of consecutive time. These controllers yield high performance levels as their fitness score is typically 1% to 4% less than the fitness score of the final solutions.

In effect, the scalability experiments have shown that parallel evolution and exchange of candidate solutions between robots enables social learning, in which a given robot benefits from another robot's experiences. Regarding odNEAT, the algorithm typically enables a significant reduction in the number of evaluations as the group increases. Hence, if intermediate controllers that operate for long periods of time can be detected and discarded via, for instance, established methods such as early stopping algorithms or racing techniques, there is the potential to accelerate and increase the performance of online evolution.

Chapter 5

From Racing to Online Hyper-Evolution

The results reported in the previous chapters have demonstrated the importance of the controller evaluation policy in online evolution of controllers for robots. Traditional online evolution algorithms employ a policy in which robots substitute controllers at regular time intervals, see Bredeche et al. (2012) for an example. While the approach is suitable for individual tasks, it can lead to incongruous group behaviour and to poor performance in collective tasks that explicitly require continuous collective coordination and cooperation between robots (Silva et al., 2015e). Contrarily to algorithms that rely on fixed evaluation periods, odNEAT allows a controller to remain active as long as it is able to solve the task. However, such approach can also be too conservative and delay the synthesis of more effective solutions (Silva et al., 2015b) by evolving intermediate controllers that can operate for a long period of time before they fail.

This chapter is divided into three parts, namely: (i,ii) two case studies, which are respectively presented in Section 5.1 and in Section 5.2, and (iii) summary and concluding remarks, which are provided in Section 5.3. In the first case study, we introduce two novel approaches to accelerate online evolution of robotic controllers in multirobot systems (Silva et al., 2016b). First, we introduce a racing approach for multirobot systems that allows individual robots to cut short the evaluation of poor controllers. Our racing approach relies on the task performance of past controllers, and therefore does not require the definition of a fixed evaluation period. Because the racing approach relies on a modified version of the non-parametric Hoeffding's bounds (Hoeffding, 1963), it can be applied to an unrestricted set of tasks and algorithms. Second, we extend racing with a population cloning approach, which enables each individual robot to clone and transmit a varying number of high-performing controllers stored in its internal population to other robots nearby, in opposition to broadcasting a single controller at a time. The underlying motivation is to effectively leverage the genetic information accumulated by multiple robots that evolve together.

We assess the performance of the different approaches in a number of simulation-based experiments. Experimental results demonstrate the ability of our proposed approaches to evolve high-performing solutions in a timely manner. Furthermore, results also show that if the evolutionary pressure is set above a certain limit, the standard version of odNEAT can, in certain conditions, outperform the racing and population cloning approaches in the long-term. This result draws attention to a novel research question: instead of using a fixed algorithm to evolve controllers online, how can we enable robots to find the best *evolutionary algorithm* to solve

a given task during the actual task execution?

In the second case study, we introduce *online hyper-evolution* (OHE). OHE is a promising approach to learning in robotic systems, as it can employ distinct algorithms to generate controllers for different tasks, and change to an algorithm better suited for controller generation if there are modifications in the requirements or environmental conditions associated with a given task. The main principle behind OHE is to use the different sources of feedback information traditionally associated with controller evaluation, namely task performance (fitness) or behaviour (novelty or diversity), in order to find suitable evolutionary algorithms online. Unlike current approaches to online evolution that rely on a single, predefined algorithm to synthesise effective controllers, OHE searches across a space of candidate algorithms and configurations in order to find effective algorithms over time.

We study three types of approaches: OHE-fitness, which uses the fitness score of controllers as the criterion to select promising algorithms over time, OHE-diversity, which relies on the behavioural diversity of controllers for algorithm selection, and OHE-hybrid, which combines diversity and fitness. In addition to their effectiveness at selecting suitable algorithms, the different OHE approaches are evaluated for their ability to *construct* algorithms for controller generation, an unprecedented approach in evolutionary robotics. Results show that OHE (i) facilitates the evolution of controllers with high performance, (ii) can increase effectiveness at different stages of evolution by combining the benefits of multiple algorithms over time, and (iii) can be effectively applied to construct new algorithms during task execution. Overall, our study shows that OHE is a powerful new paradigm that allows robots to improve their learning process as they operate in the task environment.

5.1 Online Racing and Population Cloning in Multirobot Systems

5.1.1 Background

In this section, we review the background on racing approaches in machine learning and in evolutionary computation, and we discuss current approaches to the exchange of genetic information between robots and the principles behind population cloning.

Racing

The general racing framework was originally introduced by Maron and Moore (1997) as a technique for model selection in machine learning. The key principle behind racing is to iteratively test multiple models in parallel, use the performance values of each model to discard those that are statistically inferior as soon as there is enough evidence, and then concentrate the computational effort on the remaining models. In this way, models race against each other.

Given the similarity between model selection in machine learning and parameter tuning in meta-heuristics, such as evolutionary algorithms, previous contributions have assessed how evolutionary techniques could benefit from racing procedures (Lobo, 2000; Birattari et al., 2002; Yuan and Gallagher, 2007). Haasdijk et al. (2011) studied how racing could be used to cut short the evaluation of poor controllers in online evolution. The authors used $(\mu + 1)$-online (Haasdijk et al., 2010), an encapsulated algorithm in which there is no exchange of genetic information among robots. In the native $(\mu + 1)$-online algorithm, a new controller is produced at regular time intervals, and operates for a fixed amount of time called the *evaluation period*. When the evaluation period

elapses, a new controller is synthesised and its evaluation starts. In the racing version of $(\mu + 1)$-online, the current controller is compared with those previously evaluated as it operates. If the fitness score of the controller is below a lower bound, the evaluation is aborted. The lower bound is computed based on a modified version of Hoeffding's bounds (Hoeffding, 1963) taking into account the fitness score of the worst controller in the population.

Haasdijk et al. (2011)'s study was the first demonstration of how racing techniques could be used to speed up online evolution. There are, however, a number of disadvantages regarding the authors' approach. First, algorithms such as $(\mu + 1)$-online can lead to incongruous group behaviour and poor performance in collective tasks due to the periodic substitution of controllers (Silva et al., 2015e). Second, $(\mu+1)$-online is an encapsulated algorithm, meaning that an isolated instance runs independently on each robot. In this way, $(\mu + 1)$-online does not benefit from the parallelism in multirobot systems or from exchange of genetic information between robots, which can effectively speed up online evolution (Silva et al., 2015e). Third, Haasdijk et al.'s approach was tailored to the elitist dynamics of $(\mu + 1)$-online as the lower bound of performance considers that the worst fitness score in the population does not decrease. Thus, the approach may be subject to backtracking in performance when applied to non-elitist algorithms. Finally, even with the racing technique, the combined approach still requires the experimenter to decide on the evaluation period, and is significantly sensitive to such parameter settings (Haasdijk et al., 2012).

Exchange of Genetic Information in Online Evolution

The exchange of genetic information between robots is a crucial feature in distributed, online evolutionary algorithms (Silva et al., 2015b,e). This process can be viewed as a set of *inter-robot reproduction events*, in which reproduction is implemented using other robots in the same group (Watson et al., 2002). In traditional evolutionary algorithms, selection precedes reproduction and is accomplished by having (Watson et al., 2002): (i) more-fit individuals becoming parents and supplying genes, (ii) less-fit individuals being replaced by the offspring, or (iii) by a combination of the two. In online evolution, this amounts to individual robots transmitting to neighbouring robots either part of a genome (Watson et al., 2002) or a complete genome (Bredeche et al., 2012; Silva et al., 2015e). That is, the genome is the unit in the selection process, and the population of robots is a distributed substrate in which genetic information can spread across.

In our population cloning approach, see Section 5.1.2 for a description, we take on a novel approach to the exchange of genetic information between robots. We place the selection and reproductive processes at a higher level. That is, we consider the elements in the selection process to be the internal population of each robot. A robot can therefore transmit to neighbouring robots *a copy* of any part of its population (e.g. a single genome or a set of genomes representing high-performing controllers) or of the complete population. In this way, robots have the potential to leverage the genetic information they have accumulated, and to enable a more effective knowledge transfer to solve the current task.

5.1.2 Implementing Racing and Cloning in Multirobot Systems

We propose a combined racing and cloning approach to speed up online evolution of robotic controllers in multirobot systems. The objective is to leverage the ability of racing to cut short the evolution of poor controllers, and the genetic information accumulated by individual robots evolving in parallel.

Racing

We extended odNEAT with a racing approach based on a modified version of the non-parametric Hoeffding's bounds (Hoeffding, 1963). The major advantage of Hoeffding's bounds is that it does not require any assumption regarding the underlying fitness distribution. In our racing approach, an evaluation is aborted if the controller's performance $F_{current}$ is below a lower bound L_b given by:

$$L_b = M_c(t) - 2 \cdot \xi(t) \tag{5.1}$$

$$\xi(t) = \sqrt{\frac{(F_{best} - F_{worst})^2 \cdot \log(2/\alpha)}{2 \cdot t}} \tag{5.2}$$

where $M_c(t)$ is a dynamic fitness threshold henceforth referred to as *minimal criterion* (see below), t is the current control cycle since the controller started executing, F_{worst} and F_{best} are respectively the fitness scores of the worst and best controllers in the internal population, and α is the significance level of the comparison. The minimal criterion is computed based on the fitness of the internal population. Whenever there is a change to the fitness scores of a given controller in the population (e.g. a robot receives a new controller or the fitness score of a controller is updated), $M_c(t)$ is computed as follows:

$$M_c(t) = M_c(t-1) + max(0, (v_n - M_c(t-1)) \cdot W) \tag{5.3}$$

where the value v_n corresponds to the P-th percentile of the fitness scores in the population (P is specified by the experimenter), and W is a weighting parameter that enables fine-grained control over the magnitude of the changes to the minimal criterion. Because racing approaches require a certain number of measurement points to produce reliable results (Yuan and Gallagher, 2007), we take advantage of the maturation period of odNEAT to put a lower boundary on the sample size for the racing approach. That is, racing can only abort the evaluation of a controller after the maturation period has expired.

Population Cloning

To implement our population cloning technique according to the principles described in Section 5.1.1, we adopt an approach in which two robots executing the algorithm meet, their internal populations compete, and the losing robot receives a portion of the population of the winning robot. Specifically, when two robots are in communication range, a connection link between the robots is created if none of them has been involved in a population cloning process within a predefined period of time P_c. Winner and loser are determined by comparing the $M_c(t)$ value of each robot, as defined in Equation 5.3, which is indicative of the performance of each population. The robot with the highest $M_c(t)$ value is considered the winner. Afterwards, robots execute a two-step procedure: (i) the genomes in the losing robot's population that yield a fitness score below the $M_c(t)$ of the winner robot are removed, which is conceptually similar to an extinction event, and (ii) genomes from the population of the winning robot that have a fitness score above $M_c(t)$ are injected in the losing robot.

5.1.3 Experimental Methodology

We conduct a number of simulation-based experiments in three tasks, namely standard foraging, concurrent foraging, and phototaxis. The three tasks are configured as defined in Section 4.2.1 for a group of 5 robots. We

assess the performance of three approaches: (i) standard odNEAT, henceforth called **odNEAT**, (ii) odNEAT with racing alone, which we simply refer to as **racing**, and (iii) racing plus population cloning, which we call **ppc**. For each task and each algorithm considered, we conduct 30 independent runs. Each run lasts 100 hours of simulated time. Robots operate in a square arena surrounded by walls. The size of the arena is chosen to be 3 x 3 meters. The tabu list of odNEAT is disabled because preliminary experimentation showed that it decreased performance. Other parameters and algorithmic components of odNEAT are configured as in previous experiments. Each robot executes a control cycle every 100 ms. Regarding the minimal criterion for racing, we set P to the 50th percentile of the fitness scores found in the population and $W = 1$, meaning that $M_c(t)$ amounts to the median fitness score, and $\alpha = 0.95$. In the population cloning technique, we set P_c to 100 seconds of simulated time. These parameter settings are robust to moderate variation, and were found to perform effectively in preliminary experiments. The statistical treatments are carried out using the two-tailed Mann-Whitney. When multiple comparisons are performed, we adjust p-value using the two-stage Hommel method (Hommel, 1988).

5.1.4 Experimental Results

In this section, we present and discuss the experimental results. We compare the performance of **odNEAT**, **racing**, and **ppc** in terms of performance, evolutionary dynamics, and response when the evolutionary pressure is modified.

Comparison of Performance

Figure 5.1 shows the mean fitness score of controllers throughout the simulation trials. In the two foraging tasks, **ppc** typically produces high-performing solutions in the early stages of evolution, which contributes to its superior performance. The approach consistently outperforms **racing** and **odNEAT**. In addition, the

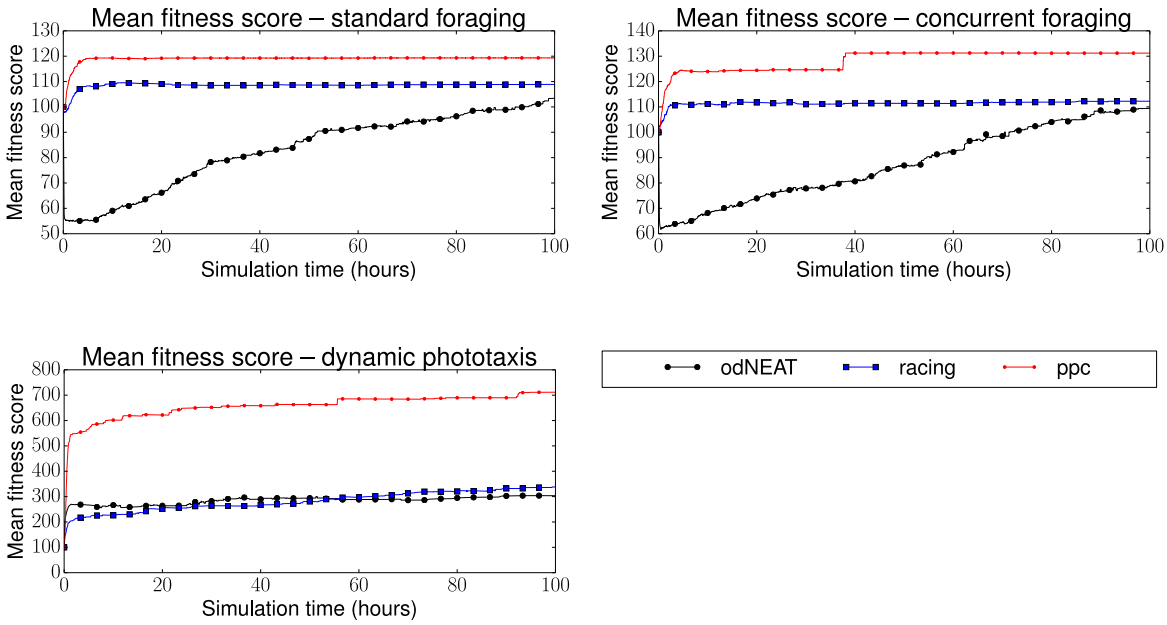


Figure 5.1: Mean fitness score of controllers throughout the simulation trials. Top: standard foraging and concurrent foraging. Bottom: dynamic phototaxis.

solution-synthesis process of racing with and without population cloning contrasts with that of **odNEAT**, which synthesises increasingly higher-performing controllers in a more progressive manner. In the dynamic phototaxis task, differences in performance between **ppc** and the remaining approaches further accentuate, which provides additional evidence regarding the benefits of racing plus population cloning.

Regarding the fitness score of the final controllers, see Figure 5.2, **ppc** leads to superior collective performance across the three tasks, which is validated by the distribution of the mean fitness of each group of robots ($\rho < 0.0001$ in the standard foraging task, $\rho < 0.01$ in the concurrent foraging task, $\rho < 0.0001$ in the dynamic phototaxis task, Mann-Whitney).

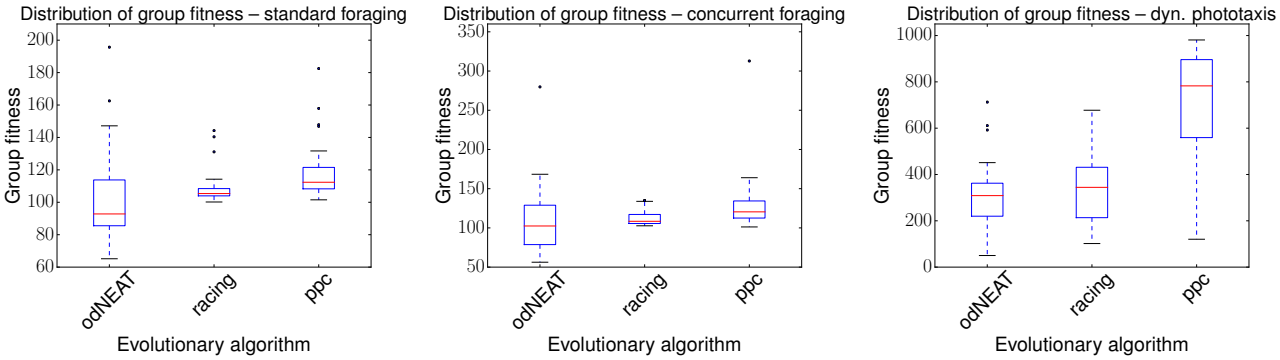


Figure 5.2: Distribution of the mean group fitness of the final controllers. From left to right: standard foraging, concurrent foraging, and dynamic phototaxis. The lower limit and the upper limit of boxplots are respectively: 65 and 147 (**odNEAT**) in the standard foraging task; 56 and 168 (**odNEAT**) in the concurrent foraging task; 50 (**odNEAT**) and 981 (**ppc**) in the dynamic phototaxis task.

In summary, our results show distinct features of racing and population cloning techniques. Comparing with **odNEAT** alone, the main benefit of **racing** is a potential speed up of evolution. The benefits of combining racing with population cloning, on the other hand, are twofold: **ppc** significantly speeds up the evolutionary process *and* leads to the synthesis of superior controllers. This result is particularly significant to online evolution because **ppc** effectively minimises the time spent assessing the quality of poor controllers, which is time spent not performing adequately at the task to which solutions are sought.

Analysis of the Evolutionary Dynamics

We first analysed how the fitness score of the final controllers vary within each group to better understand the dynamics of the approaches. To measure the *intra-group fitness variation*, we computed the relative standard deviation (RSD) of the fitness scores of each group of robots. Values close to zero indicate similar fitness values within the group. Higher values, on the other hand, indicate increasingly larger variation of fitness scores.

The distribution of RSD values is shown in Figure 5.3. Across the three tasks, **ppc** typically displays the smaller intra-group fitness variation. **odNEAT** displays significantly larger variation than **racing** ($\rho < 0.001$, Mann-Whitney). In turn, the variation of **racing** is also larger than that displayed by **ppc** ($\rho < 0.0001$ in the three tasks, Mann-Whitney). Population cloning thus leads not only to more capable controllers, but also to more consistent groups fitness-wise. Overall, results suggest a *boosting effect* on the evolutionary process and an interplay between racing and cloning towards stable collective performance (hypothesis 1). In addition, the

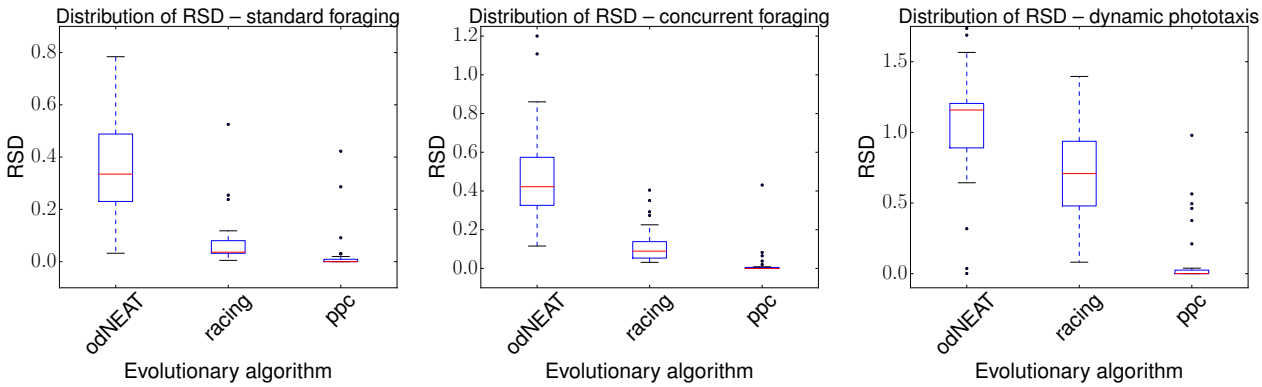


Figure 5.3: Distribution of the RSD of the final controllers. From left to right: standard foraging, concurrent foraging, and dynamic phototaxis. The lower limit of boxplots is 0.00 in the three tasks (**ppc**). The upper limit is 0.78 in the standard foraging task, 0.86 in the concurrent foraging task, and 1.57 in the dynamic phototaxis task (**odNEAT** in all cases).

high RSD values of **odNEAT** across the three tasks and the relatively higher RSD values of **racing** in the dynamic phototaxis task suggest that the performance of such approaches may be more sensitive to the task requirements and environmental pressure than the performance of **ppc** (hypothesis 2).

We conducted two sets of complementary experiments using the dynamic phototaxis task in order to verify the two hypotheses. In the first set of experiments, we removed the racing component of **ppc**, and we assessed the performance of population cloning in isolation, henceforth **ppc-rem**. In the second set of experiments, we studied the relation between evolutionary pressure and evolutionary dynamics by varying the value of the *penalty* component defined in Equation 3.5 when the light source is not in a robot's line of sight. The *penalty* was increased by a factor of 2, 5, 8, and 10, that is, to a value of -0.02, -0.05, -0.08, and -0.10 per control cycle. In this way, each controller unable to find the light source respectively executes for a period of 500, 200, 125, and 100 seconds since it started operating. These experimental setups are henceforth referred to as **p2**, **p5**, **p8**, and **p10** setups, respectively. For each configuration in each set of complementary experiments, we conducted 30 independent runs.

Isolating the Effects of Population Cloning

Figure 5.4 shows the mean fitness score throughout the simulation trials for the first set of complementary experiments. Overall, the results confirm that adding population cloning can effectively boost the evolutionary process and push towards higher-quality solutions. In addition, the median RSD of the final controllers is of 1.158 for **odNEAT**, 0.708 for **racing**, 0.375 for **ppc-rem**, and $4.17 \cdot 10^{-4}$ for **ppc**, which confirms the interplay between racing and cloning in the evolution of controllers with less disparate performance levels. The key reason for such interplay is that population cloning typically increases the fitness scores of the receiving robot's internal population and therefore the $M_c(t)$ value, which in turn contributes to further increasing the lower bound of performance of racing.

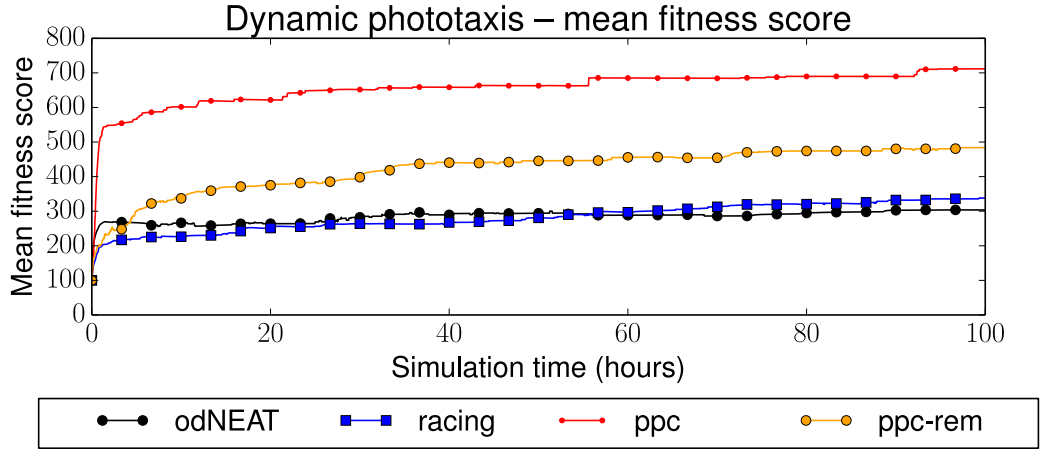


Figure 5.4: Mean fitness score of controllers throughout the simulation trials for the first set of complementary experiments (see text for details). The **ppc-rem** approach refers to population cloning without the racing component.

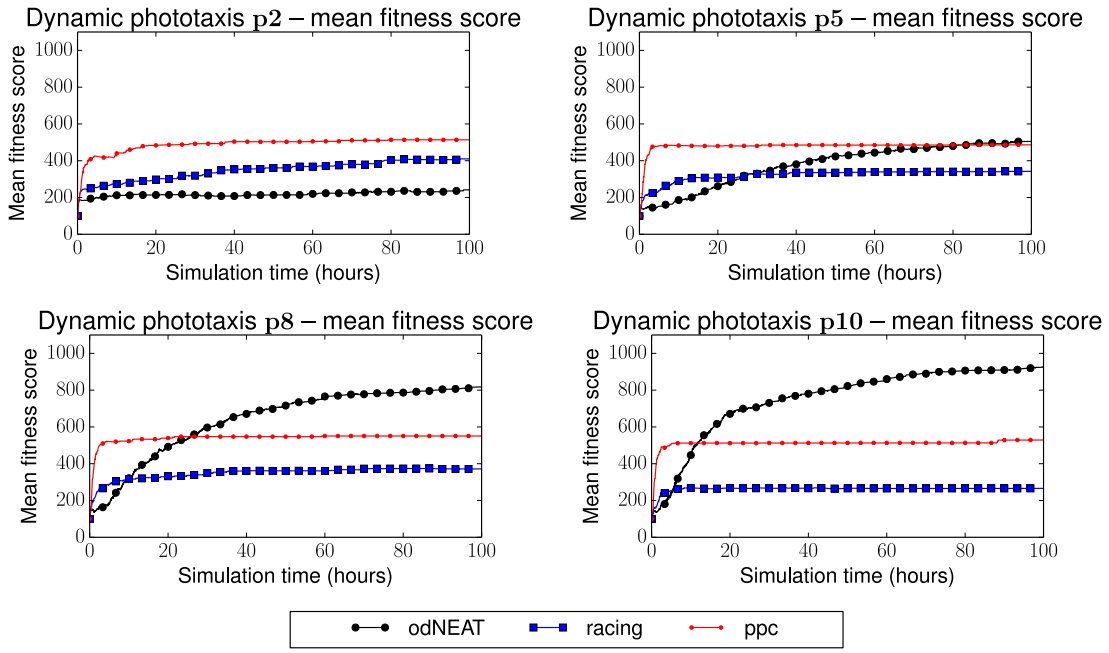


Figure 5.5: Mean fitness score of controllers throughout the simulation trials for the second set of complementary experiments (see text for details).

Modifying the Evolutionary Pressure

In terms of the second set of complementary experiments, as shown in Figure 5.5, an analysis of the mean fitness score throughout evolution shows that **ppc** achieves qualitatively similar performance levels across the setups with varying evolutionary pressures. The remaining two approaches, **odNEAT** and **racing**, yield different performance levels as the evolutionary pressure varies. From **p5** onwards, **racing** is the approach that evolves, on average, the controllers with lowest performance. The mean fitness score of the final controllers is of 410.08, 342.16, 371.75, and 265.98, respectively. **odNEAT**, on the other hand, synthesises controllers with superior performance levels as the evolutionary pressure is increased. The mean fitness score of the final controllers

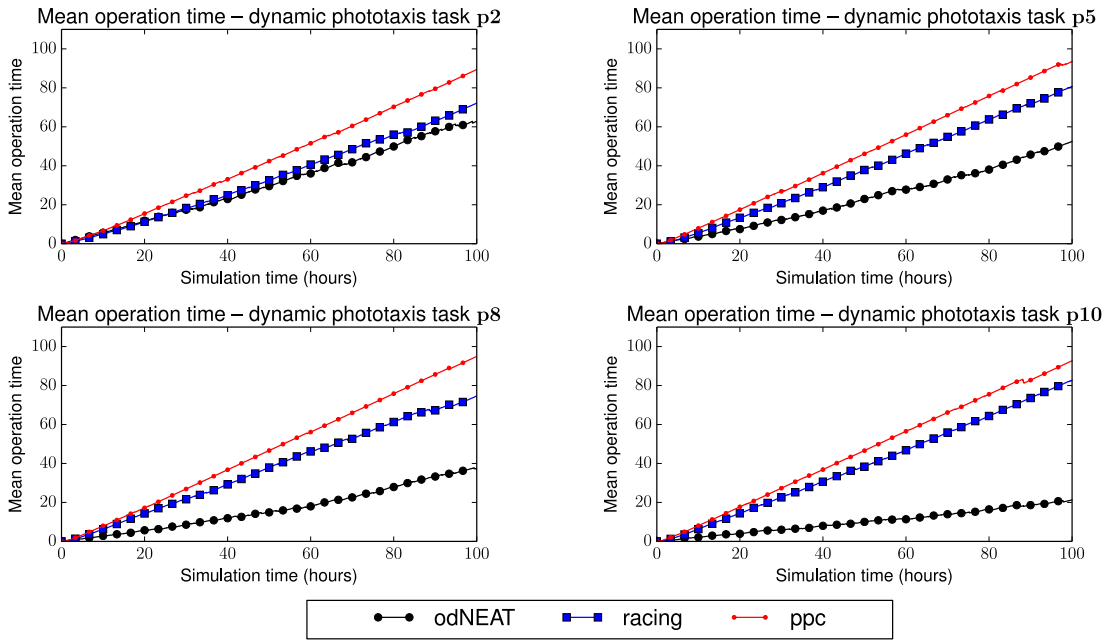


Figure 5.6: Mean operation time of controllers produced throughout the simulation trials for the second set of complementary experiments (see text for details)

varies from 241.64 in the **p2** setup to 924.24 in the **p10** setup, outperforming **ppc** in the two most demanding configurations of the dynamic phototaxis task.

One way to understand the responses of **racing** and of **odNEAT** to the evolutionary pressure is to study: (i) the operation time (age) of the controllers used by the robots during the experiments, and (ii) the number of controllers produced. The operation time of controllers relates to the robustness of solutions evolved and ability to adapt to changes in the position of the light source. Complementarily, the number of controllers produced is an indicator of the difficulty of the evolutionary process to adapt the behaviour of the robots.

Figure 5.6 shows the mean operation time of controllers during the experiments. For **ppc**, the operation time typically increases linearly, with a gentle slope, proportionally to the simulation time, which indicates that new controllers are increasingly rarely synthesised after the early stages of evolution. In effect, the final solutions synthesised by **ppc** operate, on average, up to 98 consecutive hours, which indicates that the controllers are robust and well-adapted to the task. This result, combined with the number of controllers produced, showcases the ability to quickly assess and abort the evaluation of inefficient controllers. Regarding **racing**, the algorithm displays a trend similar to that of **ppc** but yields a relatively lower mean operation time and higher number of controllers produced. This result indicates that **racing** typically requires more evaluations and more time than **ppc** to evolve stable solutions to the task. Complementarily, **odNEAT** substitutes the controller of each robot more frequently as the evolutionary pressure increases. Specifically, each robot executing **odNEAT** produces on average 24 controllers in the **p2** setup, 120 controllers in the **p5** setup, 420 controllers in the **p8** setup, and 920 controllers in the **p10** setup. This result confirms that, in this particular case, **odNEAT**'s dynamics are more sensitive than the dynamics of the racing approaches to the magnitude of the evolutionary pressure.

5.2 Introducing Online Hyper-Evolution

In the experiments reported in the previous section, we observed that the performance and relative advantages of different algorithms is conditioned on, for example, the evolutionary pressure to solve the task. Ideally, research in the field would enable the development of one or more prevalent algorithms able to effectively synthesise solutions to a large number of different tasks in a timely manner. However, as stated in the *No Free Lunch Theorem* (Wolpert and Macready, 2005), it is impossible to devise such general silver bullets given that all optimisation algorithms yield equivalent performance on average. Even if the tasks to which online evolution is applied share a common structure, an algorithm's performance is conditioned on its configuration (e.g. parameters and genetic encoding) and on the task requirements, which in turn constrains the ability for effective robot learning and adaptation. In this way, while the current quest for more efficient online evolution algorithms continues (Silva et al., 2016d), the field may ultimately need more general *mechanisms* that can combine the benefits of different algorithms.

In this section, we study a novel approach to accelerate and increase the performance of online evolution of robotic controllers. We introduce *online hyper-evolution* (OHE), in which the key principle is to use the different sources of feedback information traditionally associated with controller evaluation to find suitable evolutionary algorithms online. Contrarily to current approaches, which rely on a single, predefined algorithm to find a high-performing controller, OHE searches across a space of candidate algorithms and configurations in order to find effective algorithms over time. In this way, OHE has the potential to increase the level of generality at which online evolution can operate. OHE is distributed across a group of robots, meaning that it can leverage the inherent parallelism in multirobot systems to speed-up the search. As robots can evaluate distinct algorithms in parallel, and a given robot can make use of several algorithms throughout task execution, OHE creates the potential for a multi-trajectory search for high-performing controllers.

This section offers a comprehensive presentation and analysis of OHE. First, we propose two approaches that respectively use the fitness score and behavioural diversity of controllers produced by candidate algorithms as the criterion to select a promising algorithm at a given point in the evolutionary process. We then introduce a second class of techniques in which selection is based on a combination of fitness and behavioural diversity. We assess the different OHE techniques in two sets of experiments, namely (i) *algorithm selection experiments*, in which OHE operates over a set of three algorithms with fixed algorithmic components, including evolutionary operators for selection, crossover, and mutation, among others, and (ii) *algorithm construction experiments*, in which the choice of which components should be used is under evolutionary control. The main conclusion of our study is that OHE is an effective new paradigm because it (i) facilitates the synthesis of controllers with high performance, (ii) increases effectiveness at different stages of evolution by combining the benefits of different algorithms over time, and (iii) has an unprecedented capability to effectively construct suitable algorithms to the task to which solutions are sought.

5.2.1 Background

In evolutionary computation, there is a long history of tuning operators and parameters (Kantschik et al., 1999; Fogel, 2006). For example, in order to tune parameters during a run, several alternatives exist, such as: (i) *meta-evolution*, that is, using an evolutionary algorithm to optimise the configuration of the parameters of a task-solver

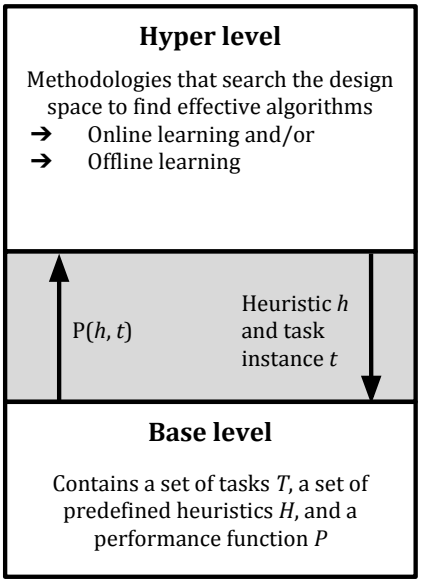


Figure 5.7: Illustration of a hyper-heuristic system.

evolutionary algorithm, (ii) *self-adaptation*, that is, using a single evolutionary algorithm that configures itself: the parameters to be self-adapted are encoded in the genome and co-evolve with the candidate solutions (Fogel, 2006), and (iii) adaptation of the parameter values according to predefined heuristic rules (Eiben et al., 2007). In effect, such approaches are related with the idea of searching a space of candidate configurations, and can thus be considered as antecedents of *hyper-heuristics* (Burke et al., 2013).

Hyper-heuristics are a search methodology intended to solve a large variety of different tasks with little or no human input. Specifically, hyper-heuristics have been described as "heuristics to choose heuristics" and as "an automated methodology for selecting or generating heuristics to solve hard computational problems", as reviewed by Ryser-Welch and Miller (2014). Hyper-heuristics systems can be represented as a two-level model (Ryser-Welch and Miller, 2014), as shown in Figure 5.7. The base level encapsulates a set of predefined heuristics for a given task, a performance function, and a given search space. The hyper level is responsible for deciding which base-level heuristic should be used to solve a given task, and may additionally generate new heuristics with a meta-heuristic search mechanism. Importantly, the hyper level can operate based on *online learning* or on *offline learning*. Similarly to online evolution algorithms, systems that employ online learning hyper-heuristics can learn directly from what they experience during task execution.

The term *hyper-heuristic* was originally introduced in Cowling et al. (2001). Afterwards, Asmuni et al. (2004) proposed a fuzzy logic system to schedule exams. Burke et al. (2007) then proposed the graph-based hyper-heuristic (GHH) approach. Graph heuristics are widely adopted methods in timetabling problems that order events not yet scheduled based on the difficulties of scheduling them into a feasible time slot. A tabu search mechanism is then used to find a suitable list of base-level heuristics.

Pillay and Banzhaf (2007) studied if and how evolutionary algorithms could be used to *evolve hyper-heuristics* for timetabling problems. The study took an approach similar to the one employed by Burke et al. (2007), except that the search process was based on a genetic algorithm instead of a tabu search. Importantly, the approach outperformed a number of previous hyper-heuristics on different test problems, thus showing the potential of

evolutionary hyper-heuristics. Other evolutionary approaches have been studied, such as applying the *messy genetic algorithm* (Goldberg et al., 1989), which processes variable-length strings, to class and exam timetabling problems (Ross and Marin-Blazquez, 2005), and combining a genetic algorithm, particle swarm optimisation, the CMA-ES algorithm, and two differential evolution algorithms under a hyper-heuristic framework to find solutions for different benchmarks (Grobler et al., 2012).

In summary, hyper-heuristics have been successfully applied to a variety of domains, including (Burke et al., 2013): (i) educational timetabling, (ii) production scheduling, (iii) workforce scheduling, (iv) constraint satisfaction, and (v) vehicle routing. Hyper-heuristics is a growing field of research with many practical applications. However, to the best of our knowledge, hyper-heuristics have not yet been assessed in evolutionary robotics domains. Further, no previous approach has been applied to the automatic construction of algorithms in evolutionary robotics. In the following section, we introduce hyper-heuristics in online evolution, which we term *online hyper-evolution*.

5.2.2 Online Hyper-Evolution

Online hyper-evolution (OHE) is a novel mechanism to increase the performance of online evolution of controllers in multirobot systems. The key principle behind OHE is that, given a set of candidate algorithms, the different sources of feedback used to assess the quality of controllers during task execution can additionally be used *at the hyper level* to select suitable algorithms over time. Below, we define the different components in OHE. Afterwards, we then detail *OHE-fitness* and *OHE-diversity*, which respectively use the fitness score and the behavioural diversity of controllers to select promising algorithms over time.

Definition

For a solution space S, a base-level online evolution algorithm can be defined as a function $a : S \rightarrow S$ that maps a population of candidate solutions to one candidate solution o called *the offspring*, which is decoded into a controller and evaluated. Let A be a set of base-level evolutionary algorithms predefined by the experimenter. During evolution, A is potentially non-static as OHE can construct new algorithms by combining the components of existing algorithms in novel ways, modify components of an existing algorithm (e.g. parameters or operators), or remove low-performing algorithms from the set. OHE is characterised by:

- A set of algorithmic components, C; $\forall a_i \in A :: C_i \subseteq C$ is a subset composed of the algorithmic components underlying a_i. The set C includes evolutionary operators for selection, reproduction, and variation of candidate solutions, and components specific to particular algorithmic configurations (e.g. racing and/or population cloning).

- A set of parameters, P, for configuring every algorithmic component in C. Parameters in P include selection probability, if crossover should be employed, crossover rate, mutation rate, among others.

- A set of tasks, T, to which solutions are sought.

- $E : S \text{ x } T \rightarrow \mathbb{R}^n$, a set of $n \geq 1$ functions that evaluate a candidate solution o at a task $t_j \in T$ according to distinct criteria (e.g. fitness and/or behavioural diversity).

- W, a workspace that maintains the evaluation score of every candidate solution o to task $t_j \in T$. Each robot in a collective maintains its own workspace. Solutions can be transmitted from one workspace to another when robots meet and exchange candidate solutions.

- $H : W$ x A x $C \rightarrow A$, a function uses the evaluation scores maintained in W to select an existing algorithm $a_i \in A$ for controller generation, modify an existing algorithm, or construct a new algorithm based on the set C of algorithmic components. That is, while the experimenter specifies a set of base-level evolutionary algorithms, the components underlying different algorithms can be combined to generate new algorithms.

Instantiating Online Hyper-Evolution

In this section, we detail two approaches to online hyper-evolution, namely OHE-fitness and OHE-diversity, which respectively use fitness and behavioural diversity as the criterion for algorithm selection.

OHE-fitness employs a single fitness-based evaluation function $e_f \in E$ that evaluates a controller by its task performance. The workspace W monitors the fitness score of controllers in the population of the robot, and keeps track of which algorithm $a_i \in A$ generated which controller. To that end, each genome is augmented with an integer identifier of the algorithm a_i that has originated it, where $i \in \{1..N\}$ and N is the number of algorithms in A. When genomes are exchanged between robots, the fitness score of the corresponding controller and the identifier of the algorithm are also sent to the receiving robot. Thus, OHE-fitness can operate in a distributed and decentralised manner.

OHE-fitness intervenes in the evolutionary process when it is necessary to generate a new controller for a robot. Upon initialisation, that is, when the robot first starts executing, the function H randomly selects a base-level algorithm for controller generation. In subsequent interventions, H selects an algorithm a_i randomly with a small probability p_{rs} or proportionally to the mean fitness score of the controllers produced by a_i. After the algorithm selection step, evolution proceeds as usual, with the configuration of the chosen algorithm during controller synthesis and evaluation. **OHE-diversity** employs a diversity-based evaluation function $e_d \in E$ that scores a controller according to behavioural diversity. Specifically, the behaviour is iteratively characterised at every control cycle of the robot (see below for details). When a new controller for the robot is needed, the function H selects a candidate algorithm $a_i \in A$ randomly with a small probability p_{rs}, or proportionally to the mean degree of behavioural diversity of the controllers generated by a_i.

In H, the behavioural diversity score of a controller x is computed as the mean distance to its k-nearest neighbours in the population of the robot (Mouret and Doncieux, 2012). In the following section, we describe the two components required for behaviour diversity computations, namely the *behaviour characterisation* and the distance metric *dist* that quantifies differences between behaviours.

Distributed State Count Characterisation

An effective task-dependent behaviour characterisation is usually the product of extensive trial-and-error experimentation (Silva et al., 2016d). We thus rely on a generic, task-independent characterisation called *Distributed State Count* (Silva et al., 2016c). The key principles behind the Distributed State Count characterisation are: (i) to define states based on the sensor readings and actuation values at every control cycle during controller evaluation, (ii) to compute the number of times a controller visited a given state, and (iii) when robots meet and

exchange genomes, to propagate states counts along with genetic information. For algorithmic efficiency, each characterisation is represented as a map from states to counts. In Algorithm 4, we describe the computation of the characterisation as independently executed by each robot during controller evaluation.

Algorithm 4 Pseudo-code of the Distributed State Count characterisation for a robot r.

```
controller_r ← assign_as_controller(genome, r)
m ← Map < Integer, Integer >
alg_id ← id_of_current_algorithm
while controller_r is under evaluation do
    execute_control_cycle()
    ϑ_r ← read_state(r)
    ϑ'_r ← discretise(ϑ_r)
    h ← hash(ϑ'_r)
    m[h] ← m[h] + 1
    if broadcast? then
        send(genome, alg_id, m, robots_in_range)
    end if
end while
```

The function $read\_state(r)$ retrieves the vector of sensor readings and actuation values ϑ_r for robot r:

$$\vartheta_r = \{\mathbf{s}(r), \mathbf{a}(r)\} \tag{5.4}$$

where $\mathbf{s}(r)$ and $\mathbf{a}(r)$ are the sensor readings and actuation values. The function $discretise(\vartheta_r)$ computes the vector ϑ'_r by independently normalising each component in ϑ_r, and rounding the resultant value to the nearest integer:

$$\vartheta'_{i,r} = \left\| \frac{\vartheta_{i,r} - \vartheta_{i,min}}{\vartheta_{i,max} - \vartheta_{i,min}} \cdot (B-1) \right\|, i \in [1, N] \tag{5.5}$$

where $\vartheta_{i,max}$ and $\vartheta_{i,min}$ are respectively the maximum and minimum value of the i-th sensor/actuator, B is the number of discrete bins, and N is the length of ϑ_r and ϑ'_r. In this way, the parameter B is used to define the resolution of the behaviour characterisation. Finally, the function $hash(\vartheta'_r)$ employs Jenkins' one-at-a-time function to hash the vector ϑ'_r into a single integer in order to improve the space- and time- complexity of the algorithm. The behavioural distance $dist$ between two characterisations m_1 and m_2 is given by a modified version of the Bray-Curtis dissimilarity, a statistic used to quantify the difference between samples of abundance data:

$$dist(m_1, m_2) = \frac{\sum_{i \in m_1 \cap m_2} |m_1[i] - m_2[i]| + \sum_{i \in m_1 - m_2} m_1[i] + \sum_{i \in m_2 - m_1} m_2[i]}{\sum_{i \in m_1} m_1[i] + \sum_{i \in m_2} m_2[i]} \tag{5.6}$$

Driving the Search Process

OHE-fitness and OHE-diversity can aid evolution by changing the algorithm used to search for promising solutions. However, the performance of OHE is fundamentally tied to how the search is guided at the *base level*. That is, the selective pressure and effectiveness of OHE influences and is influenced by the selective pressure of the individual algorithms at base level, giving rise to a *multi-level* selective pressure, see Figure 5.8.

To investigate the multi-level selective pressure within OHE, we study **OHE-diversity+fit** and **OHE-diversity+div**, which respectively represent OHE-diversity with fitness-based selection and diversity-based

selection driving the base-level algorithms; **OHE-fitness+fit** and **OHE-fitness+div**, that is, OHE-fitness operating with fitness-based selection and diversity-based selection at the base level. Behavioural diversity methods are an exploration procedure in the sense that they encourage a more expansive search. Complementarily, fitness-based algorithms are an exploitation procedure as they typically focus on increasingly narrow regions of the search space. In this way, the different variants of OHE-fitness and of OHE-diversity allow us to assess the relative merits of encouraging exploration and/or exploitation at different levels.

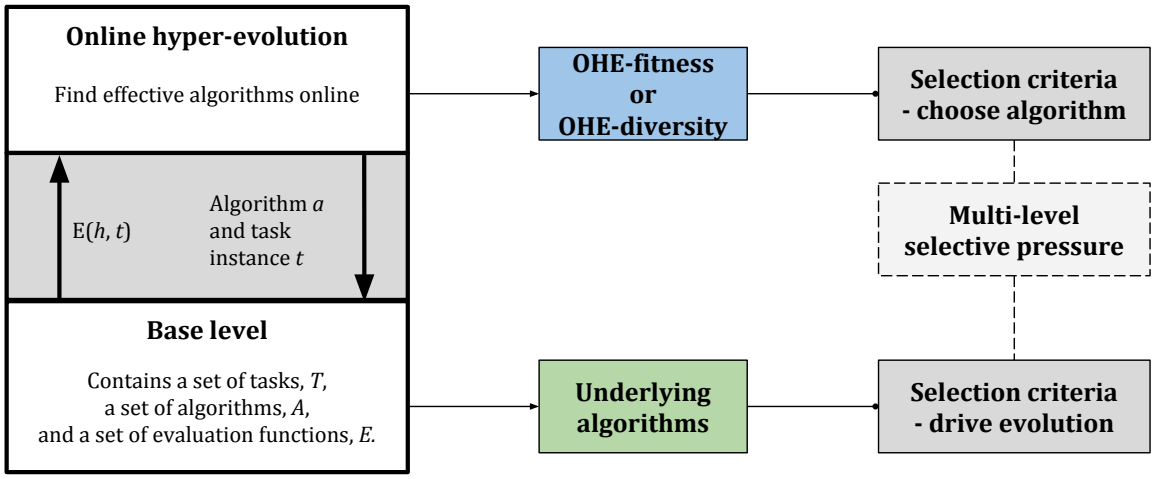


Figure 5.8: Illustration of the multi-level selective pressure in online hyper-evolution.

Experimental Setup

Given its generality, OHE can be implemented over a number of different algorithms. We use the three algorithms assessed in the previous chapter as the base-level algorithms, namely **odNEAT**, **racing**, and **ppc**. We assess the performance of the seven approaches (three base-level algorithms and four OHE variants) in the same set of tasks considered in the previous section, namely: (i,ii) two foraging tasks with different complexity, and (iii) five configurations of the dynamic phototaxis task with varying evolutionary pressures (**p1**, **p2**, **p5**, **p8**, and **p10**).

For each task variant and each algorithm considered, we conduct 30 independent runs. Each run lasts 100 hours of simulated time. For OHE approaches, p_{rs} is set to 0.10. OHE approaches with a behaviour diversity component use a k value of 5 nearest neighbours. The number of bins is set to $B = 10$ bins. k and B were tuned empirically. These parameter settings are robust to moderate variation, and were found to perform effectively in preliminary experiments.

In addition to the quality of the controllers evolved, another relevant aspect is how the different algorithms explore the behaviour space, individually and with respect to each other. To visualise how the different approaches traverse the behaviour space, we use *Sammon's nonlinear mapping* (Sammon Jr., 1969). Note that this application of Sammon's mapping differs from that of previous experiments, in which Sammon's mapping was used for visualisation and analysis of the genotypic search space.

5.2.3 Experimental Results

In this section, we assess our proposed OHE approaches. Firstly, we compare the performance of the most straightforward OHE approach, namely **OHE-fitness+fit**, to the performance of each individual evolutionary

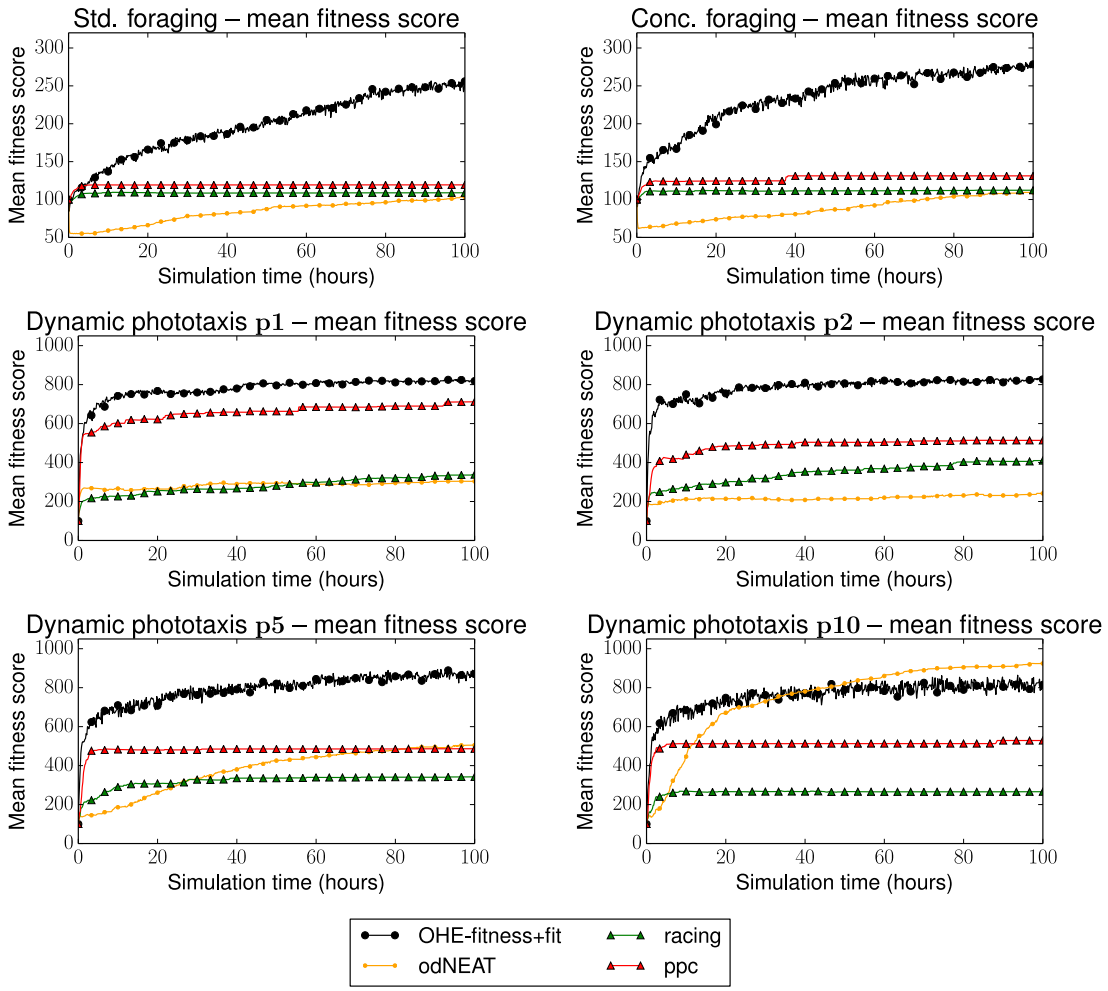


Figure 5.9: Mean fitness score of controllers evolved by base-level algorithms and by **OHE-fitness+fit** throughout the simulation trials of the standard foraging and concurrent foraging tasks (top), and four variants of the dynamic phototaxis task (centre and bottom).

algorithm. We then compare the different OHE approaches in terms of task performance and exploration of behaviour space.

OHE-fitness+fit vs. Base-level Algorithms

The mean fitness score of controllers throughout the simulation trials is shown in Figure 5.9 for the two foraging tasks, and for four variants of the phototaxis task (**p1**, **p2**, **p5**, and **p10** setups). Overall, results show that **OHE-fitness+fit** typically yields superior performance and that it is able to synthesise effective solutions in the early stages of evolution. In the two foraging tasks, **OHE-fitness+fit** typically yields performance levels superior to those of the individual algorithms, and the highest-performing individual algorithm is **ppc**. Differences between the mean group fitness of the final controllers evolved by **OHE-fitness+fit** and that of those evolved by other algorithms are statistically significant across every comparison ($\rho < 0.0001$, Mann-Whitney).

In the dynamic phototaxis task, **OHE-fitness+fit** is still the highest-performing approach, but the differences in performance between **OHE-fitness+fit** and the individual algorithms is dependent on the evolutionary pressure. In the least demanding configuration (**p1** setup), **OHE-fitness+fit** significantly outperforms

odNEAT and **racing** ($\rho < 0.0001$, Mann-Whitney), and **ppc** is the algorithm that gets closest to the performance levels of **OHE-fitness+fit**. In the **p2** and **p5** setups, **OHE-fitness+fit** also outperforms every other algorithm ($\rho < 0.001$ and $\rho < 0.0001$, Mann-Whitney). However, as the task becomes even stricter, the performance of **odNEAT** furthermore increases, and the algorithm is able to outperform **OHE-fitness+fit** in the final part of the **p10** setup ($\rho < 0.0001$, Mann-Whitney).

Algorithm Selection Analysis

To investigate the dynamics of **OHE-fitness+fit**, we analysed the base-level algorithms used by the approach. Throughout this section, we refer to the selection and execution of a given algorithm by OHE as an *algorithm instance*.

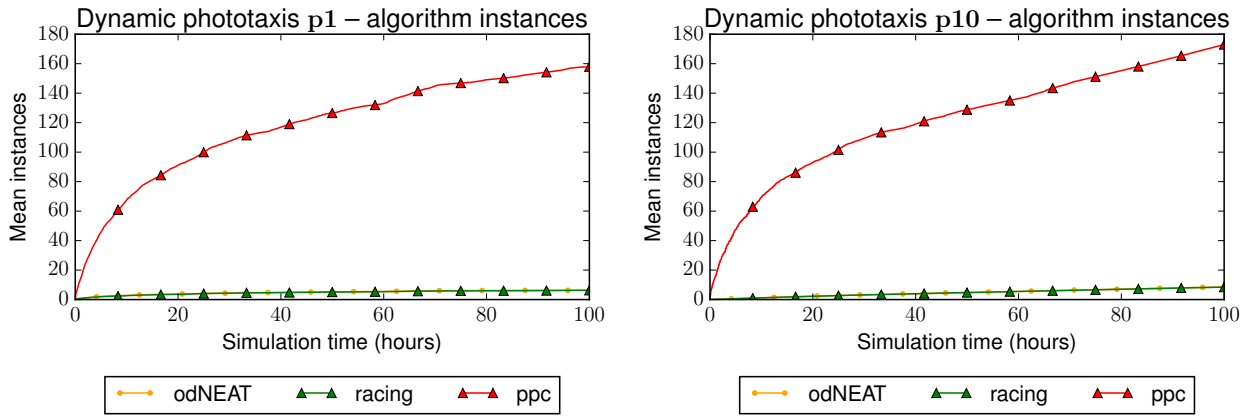


Figure 5.10: Mean number of algorithm instances used by **OHE-fitness+fit** throughout evolution in the **p1** setup and in the **p10** setup.

Figure 5.10 shows the mean number of algorithm instances used by **OHE-fitness+fit** in the **p1** setup and in the **p10** setup. Remaining tasks yield similar trends. Throughout evolution, **OHE-fitness+fit** selects **ppc** significantly more often than the other two algorithms to synthesise effective controllers ($\rho < 0.0001$, Mann-Whitney). Specifically, **ppc** is used to generate, on average, 90% or more of the controllers evaluated during the search process. For example, in the **p1** and **p10** setups, **ppc** is chosen 158 times and 173 times per robot, while **odNEAT** and **racing** are each chosen approximately six times per robot and nine times per robot, respectively. Importantly, **ppc** is typically used more intensively in the early stages of evolution, thus indicating that it helps to boost the evolutionary process and push towards higher-quality solutions.

Overall, our results illustrate the potential of **OHE-fitness+fit**, namely that it can effectively combine the benefits of different algorithms to speed up and increase the performance of the evolutionary process. In the following section, we study if and how the performance of OHE changes with respect to how the search is guided at the hyper level and at the base level.

Multi-level Selective Pressure

In this section, we analyse the performance and behaviour of different OHE variants, namely **OHE-fitness+fit**, **OHE-fitness+div**, **OHE-diversity+fit**, and **OHE-diversity+div**. Figure 5.11 shows the mean fitness score

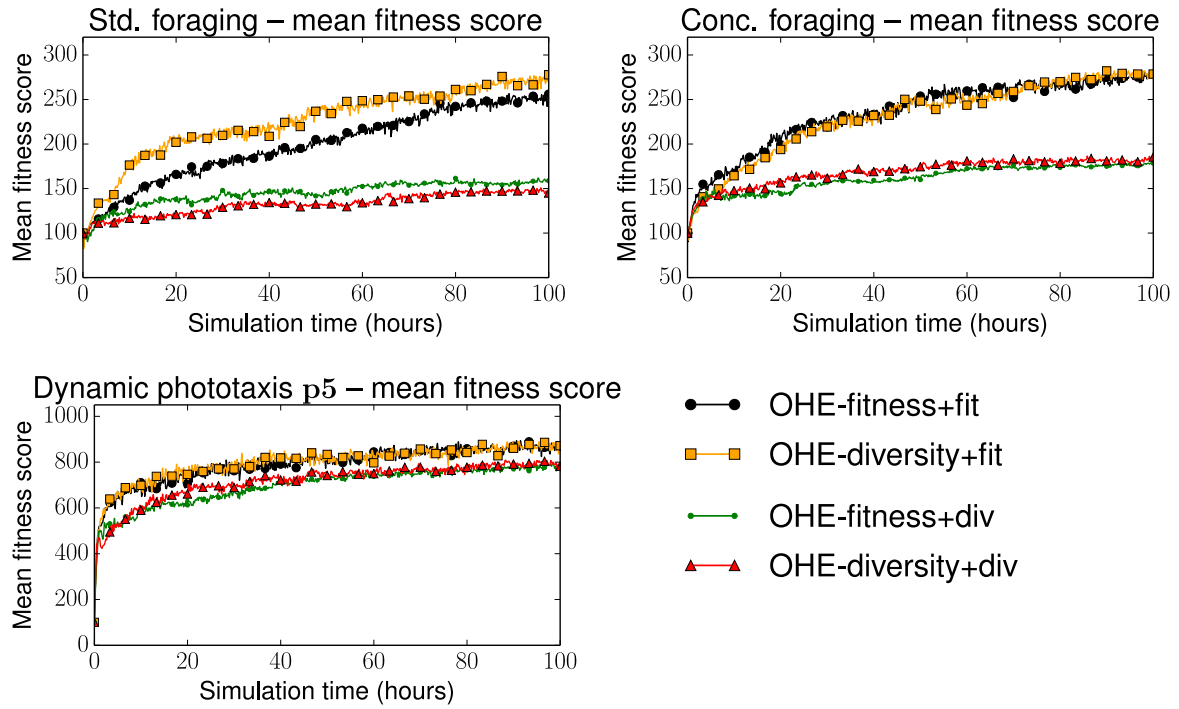


Figure 5.11: Mean fitness score of controllers evolved by the different OHE techniques throughout the simulation trials of: (i) the standard foraging and concurrent foraging tasks (top), and (ii) in the **p5** setup (bottom).

throughout evolution for the two foraging tasks and for the dynamic phototaxis **p5** setup. Remaining variations of the dynamic phototaxis task yield comparable results. Firstly, results show that driving evolution towards behavioural diversity at the base level is detrimental to the performance of OHE. In this respect, differences between the fitness scores of final controllers synthesised by fitness-based evolution and those synthesised by diversity-based evolution are significant in the two foraging tasks ($\rho < 0.0001$), in the **p1** setup ($\rho < 0.05$, scores not shown in Figure 5.11), and in the **p5** setup ($\rho < 0.005$). Secondly, with respect to **OHE-fitness+fit** and **OHE-diversity+fit**, the two approaches yield comparable performance levels in all tasks, although there is a slight advantage in favour of **OHE-diversity+fit** in the standard foraging task

Figure 5.12 shows the mean amount of time per robot that each algorithm was used by **OHE-fitness+fit** and **OHE-diversity+fit** in the different tasks. This statistic is given by the mean cumulative operation time of the controllers produced by each method. While the search mechanisms underlying the different levels in OHE can exert significant influence on the evolutionary path over time, **OHE-fitness+fit** and **OHE-diversity+fit** adopt similar strategies and execute the base-level evolutionary algorithms for comparable amounts of time in the majority of the tasks. Another commonality between the two methods is that when **odNEAT** and the combined racing and population cloning approach (**ppc**) are selected to synthesise controllers, the fitness score of controllers increases on average from approximately 20% to 160%. When **racing** is selected, on the other hand, fitness scores typically decrease by up to 19%. That is, the performance of **OHE-fitness+fit** and **OHE-diversity+fit** is based on an effective combination of **odNEAT** and **ppc** throughout evolution. The key result is thus that different algorithms are important at different phases of evolution. This indicates that OHE is more than an effective algorithm selection approach: OHE can increase effectiveness and boost performance at different stages of evolution.

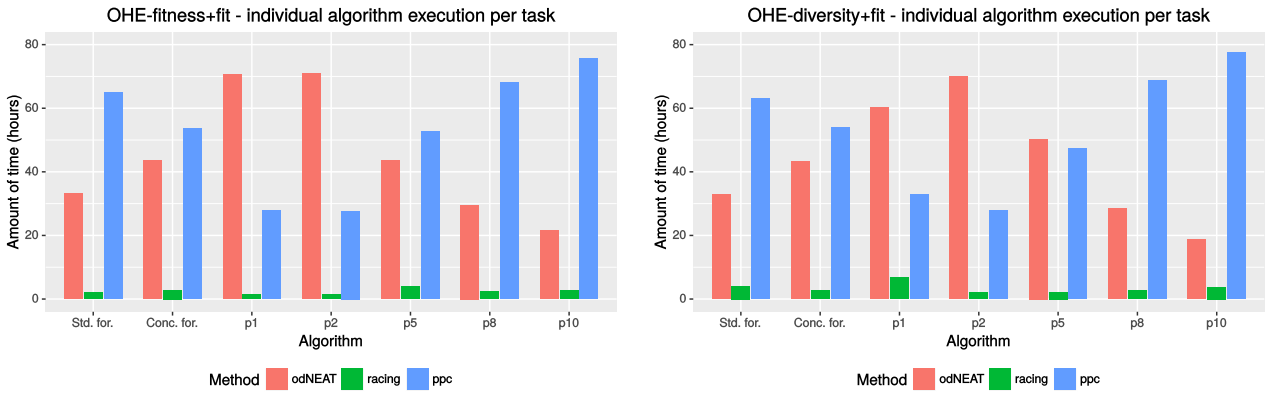


Figure 5.12: Mean amount of time per robot that each algorithm was used by **OHE-fitness+fit** (left) and **OHE-diversity+fit** (right).

Behaviour Space Analysis

To further assess the dynamics of the OHE approaches, we analysed how they traverse the behaviour space using Sammon's mapping (Sammon Jr., 1969). The distance in the high-dimensional space δ_{i_j} between the behaviour of two controllers i and j is given by the count of states (number of states and respective cardinality) in which the behaviours differ. The distance between two points in the two-dimensional space is their Euclidean distance. Note that **OHE-fitness+fit** does not make use of behavioural diversity during evolution.

The three mappings in Figure 5.13 show how the different approaches compare with one another in terms of behaviour space exploration (**p5** setup, remaining tasks yield similar results). To obtain a clear visualisation and a representative selection of behaviours, we mapped the 250 most behaviourally different controllers produced by each of the four approaches. The area of each point in the two-dimensional space is set proportionally to the fitness score of the corresponding controller. Behaviours belonging to high-fitness regions (fitness $> 75\%$ of maximum fitness, approximately 750) are marked with a grey square in the centre of the corresponding point. The error value is $E_m = 0.025$ for the first two mappings, and 0.027 for the third mapping, which indicates that the distances between behaviours are well-preserved.

The first mapping is shown in Fig 5.13(top left), and compares the two OHE-fitness variants, **OHE-fitness+fit** and **OHE-fitness+div**. Given its ability to guide evolution towards diversity, **OHE-fitness+div** naturally performs a more uniform exploration of the behavioural search space than **OHE-fitness+fit**. The second mapping compares the two approaches that exploit fitness-based evolution and diversity-based evolution in opposite ways. Comparing with **OHE-fitness+div**, the **OHE-diversity+fit** approach is able to both cover more regions of the behaviour search space, *and* to find higher-performing solutions in those regions, see behaviours located in the right half of Figure 5.13 (top right). Complementarily to the results in Figure 5.11, which show that **OHE-diversity+fit** is typically the highest-performing approach, the behaviour space exploration analysis indicates that it is beneficial to the performance of OHE if the hyper-level search is driven towards behavioural diversity, and the base-level search is driven towards higher fitness regions. In effect, as shown in the third mapping in Fig 5.13, **OHE-diversity+fit** also explores more regions of *high-performing* solutions than **OHE-diversity+div**, which is driven towards diversity in both levels.

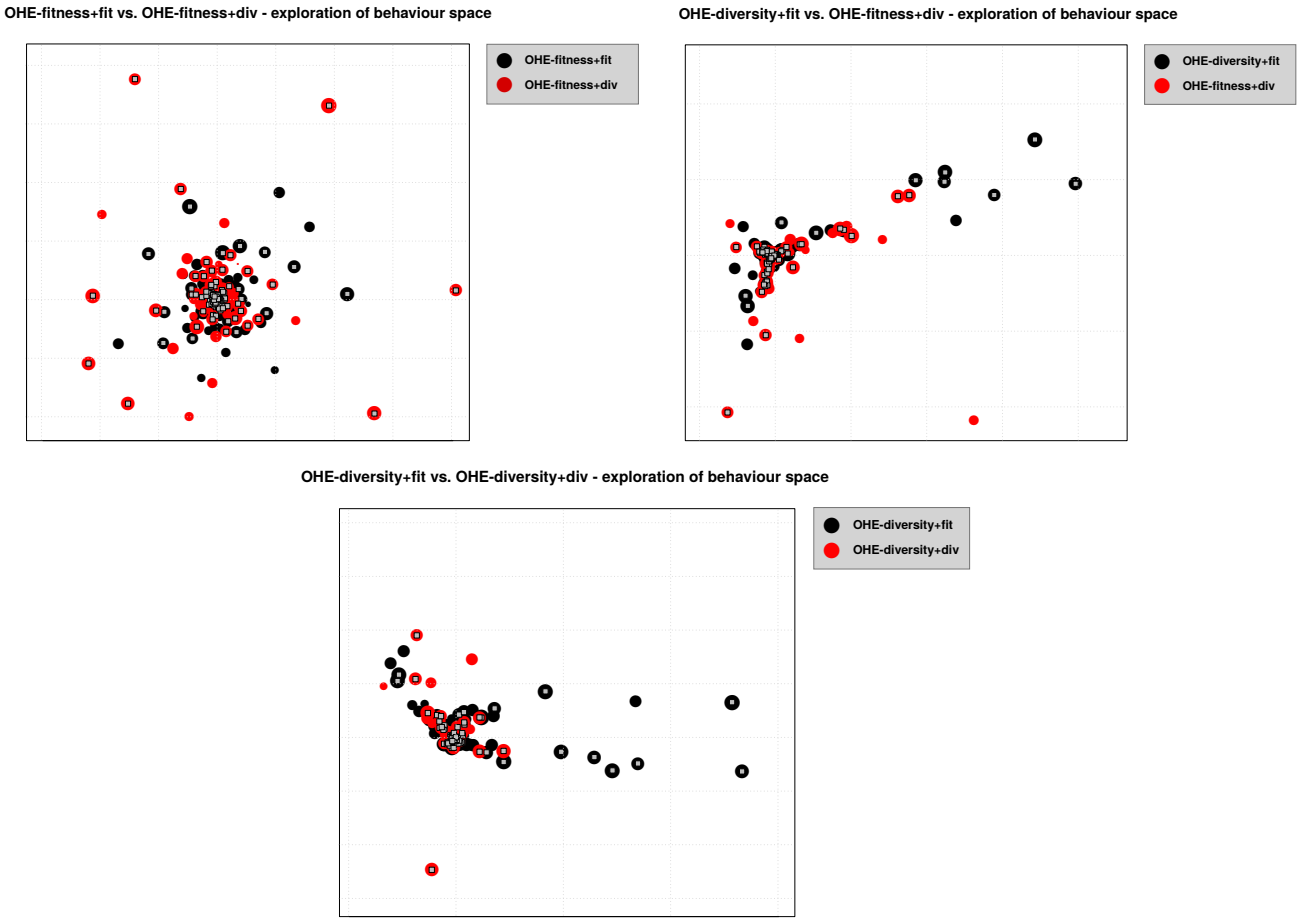


Figure 5.13: Sammon's mapping for the **p5** setup: (top left) **OHE-fitness+fit** vs. **OHE-fitness+div**, (top right) **OHE-diversity+fit** vs. **OHE-fitness+div**, and (bottom) **OHE-diversity+fit** vs. **OHE-diversity+div**. Behaviours in high-fitness regions are marked with a grey square.

Overall, the analyses in this section show that the OHE approaches have different dynamics both in terms of how they use the base-level algorithms, and of how they traverse the behaviour space. The four approaches consistently yielded high-performance across the tasks, and **OHE-diversity+fit** was found to be the most effective one in the current experimental settings.

5.2.4 Improving the Hyper-Level

The experimental results presented in the previous section showed that driving evolution towards behavioural diversity at the base level is often detrimental to the performance of OHE. The results furthermore demonstrated that **OHE-diversity+fit** is a promising approach as it can simultaneously push towards diverse and high-performing behaviours. However, one potential issue with behavioural diversity-based approaches is that they may spend a significant amount of effort exploring regions of the behaviour space that are unrelated with the task objective, or may diverge if the behaviour space is vast (Doncieux and Mouret, 2014).

In this section, we investigate if and how the hyper-level can be improved by adding a fitness component to the diversity-based search in order to circumvent the aforementioned issues. This novel class of techniques is called *OHE-hybrid*. In OHE-hybrid, the set E of evaluation functions is composed of e_f and e_d, as used

separately in OHE-fitness and OHE-diversity (see Section 5.2.2 for details). As in the previous experiments, algorithms are selected proportionally to the mean score of the controllers they produced. At the hyper level of OHE-hybrid, the score of a controller is given by combining behavioural diversity and fitness as follows:

- **LS-50** Upon algorithm selection, the score of a controller is given by the weighted sum of the normalised fitness score and of the normalised behavioural diversity score (Cuccu and Gomez, 2011). The two scores have the same weight: 50%.

- **LS-75**[1] Similar to **LS-50**, but the behavioural diversity score has a weight of 75%, while the fitness score has a weight of 25%.

- **MCDS** Minimal criteria diversity search, which is inspired by minimal criteria novelty search (Lehman and Stanley, 2010) with non-static criterion (Gomes et al., 2012). Controllers below a minimal criterion receive a score of zero, while controllers above the criterion receive their normal behavioural diversity score. The minimal criterion is progressive and, similarly to **racing**, corresponds to the 50-th percentile of the fitness scores in a robot's population (no decreases in the criterion are allowed).

- **NSGA** Multiobjective combination of behavioural diversity and fitness using the NSGA-II algorithm (Deb et al., 2002). After NSGA-II's nondominated sorting and crowding-distance assignment, controllers belonging to the first nondominated front are selected, scored according to their crowding distance (all controllers are on the same front, and therefore share the same rank in NSGA-II), and used seamlessly in OHE.

In addition to combining behavioural diversity and fitness, we conduct experiments using regular **OHE-fitness+fit** and **OHE-diversity+fit**. In the forthcoming experiments, these methods are referred to as **Fit** and **Div** for consistency with respect to the approaches listed above.

Experimental Setup

We have devised two experimental setups to assess the relative advantages and disadvantages of the different approaches, respectively called the *algorithm selection* experiments and the *algorithm construction* experiments. In the algorithm selection experiments, OHE can change the algorithm used to search for promising solutions (**odNEAT**, **racing**, **ppc**), as in the experiments described in the previous section. In the algorithm construction experiments, we take a step towards the construction of complete algorithms during task execution. We place under evolutionary control which of the main algorithmic components underlying the base-level algorithms should be used: (i) crossover, (ii) mutation, which includes the addition of new neurons, the addition of new connections, and the optimisation of connection weights, (iii) niching, namely speciation and fitness sharing, (iv) racing, and (v) racing with population cloning. There are thus a total of 32 possible algorithmic configurations, and a significantly larger search space for OHE to traverse when comparing with the algorithm selection experiments.

In order to systematically compare the different approaches, we resort to the following measures:

[1] We experimented with weights up to 95% for the diversity score. Results were found to be qualitatively similar to when the diversity score weights 75%.

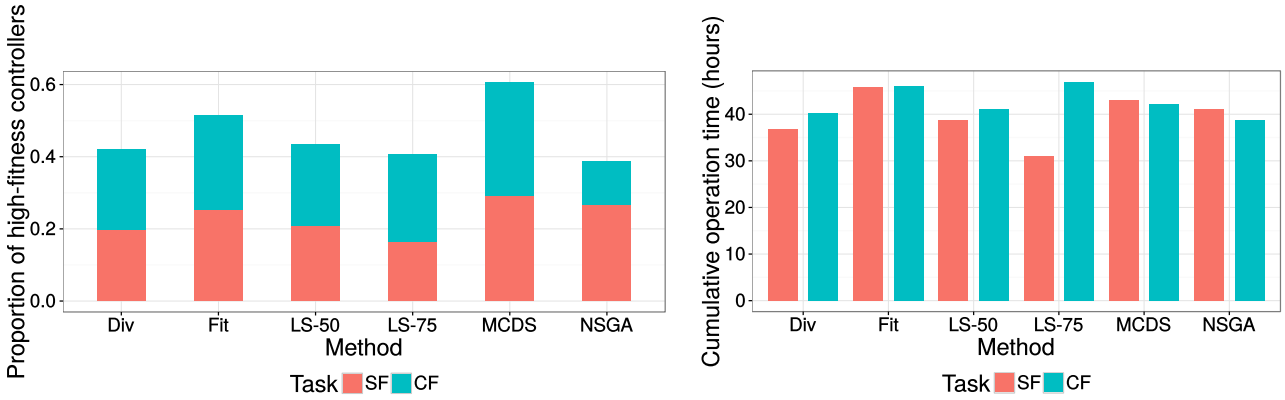


Figure 5.14: Analysis of successful runs in the algorithm selection experiments. Left: proportion of high-fitness controllers evolved. Right: cumulative operation time of high-fitness controllers. SF and CF stand for standard foraging and concurrent foraging, respectively.

- Success rate of the different methods in a given setup. The success rate is defined as the portion of runs in which at least one robot evolved a high-performing controller to the task. We have empirically determined that high-performing controllers are respectively characterised by a fitness score of approximately 250 or above in foraging tasks, and a fitness of 750 or above in the dynamic phototaxis tasks. Success rates are compared using the two-tailed Fisher's exact test.
- Proportion of high-fitness controllers produced in a given run or setup. Only successful runs are considered in this statistic.
- Operation time of high-fitness controllers, which may vary from one high-fitness controller to another because of dynamic evaluation times of the base-level algorithms.

Algorithm Selection Experiments

Table 5.1 lists the success rates for the two foraging tasks. Results show that **LS-50** is the approach with the highest success rate in the two tasks (45/60), followed by **Div** (42/60), and **LS-75** and **MCDS** (40/60). **Fit** and **NSGA** yield the lowest success rates in the two foraging tasks. The low success rate of **NSGA** is noteworthy, which suggests that the algorithm may struggle to balance exploration and exploitation effectively in this setup.

Table 5.1: Algorithm selection experiments. Success rates for the standard foraging task and for the concurrent foraging task. In the different phototaxis tasks, all methods had a success rate of 30/30.

Task	**Div**	**Fit**	**LS-50**	**LS-75**	**MCDS**	**NSGA**
Std. foraging	16/30	14/30	19/30	14/30	15/30	12/30
Conc. foraging	26/30	23/30	26/30	26/30	25/30	22/30

In the dynamic phototaxis tasks, the proportion of high-fitness controllers typically varies from 0.70 to 0.80 for the different methods. That is, all approaches generate a similar, large portion of high-fitness controllers to the tasks. In the foraging tasks (see Figure 5.14), however, **MCDS** stands out as the method that, when successful, is: (i) more biased towards high-fitness regions, and (ii) more consistent in terms of the proportion

Table 5.2: Algorithm construction experiments. Success rates for the standard foraging task and for the concurrent foraging task. In the **p1**, **p2**, **p5**, and **p8** setups, all methods have a success rate of 30/30. In the **p10** setup, **Fit** and **NSGA** have a success rate of 29/30, while other methods have a rate of 30/30. The "vs. algorithm selection exp." field denotes the differences in the success rates with respect to the algorithm selection experiments.

Task	**Div**	**Fit**	**LS-50**	**LS-75**	**MCDS**	**NSGA**
Std. foraging	13/30	17/30	9/30	12/30	16/30	10/30
(vs. algorithm selection exp.)	-3	+3	-10	-2	+1	-2
Conc. foraging	20/30	23/30	19/30	25/30	21/30	24/30
(vs. algorithm selection exp.)	-6	0	-7	-1	-4	+2

of high-fitness individuals evolved in the two foraging tasks. In terms of the cumulative operation time of high-fitness controllers, **Fit** excels and is closely followed by **MCDS**. **LS-75** appears to be sensitive to the task requirements both with respect to the proportion of high-fitness controllers evolved (0.17 in the standard foraging task vs. 0.24 in the concurrent foraging task), and in the operation time of high-performing controllers.

Overall, the results presented in this section show that the hyper-level can be improved via the addition of mechanisms that balance diversity and fitness, but that there are trade-offs to consider. For example, **MCDS** has a slightly lower success rate than **Div** but, in the successful runs, is able to evolve a larger proportion of high-fitness controllers ($\rho < 0.01$), and such controllers yield higher operation times. In the following section, we assess if and how these results hold in a more challenging scenario, the algorithm construction experiments.

Algorithm Construction Experiments

Table 5.2 lists the success rates for the different configurations of the algorithm construction experiments. **LS-50**, one of the most effective approaches in the algorithm selection experiments, evolves high-performing controllers in 17 runs less in the algorithm construction experiments. In the standard foraging task, differences between the success rate in the algorithm construction experiments and in the algorithm selection experiments are statistically significant ($\rho < 0.05$, Fisher's exact test). The differences in success rates between **LS-50** and **LS-75** furthermore confirm that the performance of linear scalarisation is highly dependent on the weighting (Cuccu and Gomez, 2011). That is, while linear scalarisation is conceptually simple to implement, and can be used in conjunction with any algorithm and objectives, it is a limiting approach with respect to scaling OHE to constructing complete algorithms online.

Similarly to **LS-50**, the success rates of **Div** are lower in the algorithm construction experiments (high-performing controllers were evolved for 9 runs less). This result suggests that methods with a strong diversity component may suffer in higher-dimension search spaces. This observation is complemented by the results of the **Fit** method, whose success rates only decreased in one run of the **p10** setup, and actually increased slightly in the standard foraging task. Finally, **NSGA** yields similar success rates in the algorithm construction experiments and in the algorithm selection experiments. **MCDS** has a slightly lower success rate in the concurrent foraging task, yet remains as one of the methods with highest overall success rate. In effect, as shown in Figure 5.15, **MCDS** is the only method that consistently achieves a comparatively high cumulative operation time in the two foraging tasks.

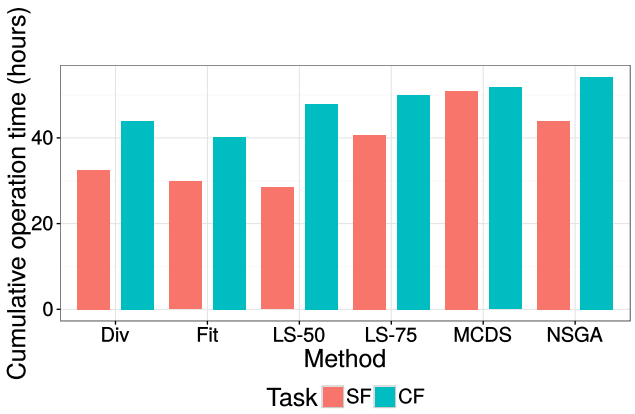


Figure 5.15: Analysis of successful runs in the algorithm construction experiments. Mean cumulative operation time of high-fitness controllers. SF and CF stand for standard foraging and concurrent foraging, respectively.

5.3 Summary

In this chapter, we studied different approaches to accelerate and increase the performance of online evolution of controllers in multirobot systems. In the first case study, we assessed: (i) a racing technique, and (ii) a population cloning technique. We showed the benefits of our racing approach, and that population cloning allows evolution to effectively leverage the genetic information accumulated by each individual robot. The combined racing and population cloning approach typically yielded: (i) the highest task performance in terms of the fitness score, (ii) the fastest evolution of effective solutions to the task, (iii) the most consistent and stable group-level performance, and (iv) the highest degree of robustness as the evolutionary pressure to solve the task increases. However, if the evolutionary pressure is set above a certain limit, the standard version of odNEAT can, in certain conditions, display superior performance in the long-term. One key research question is therefore how to enable robots to find the best evolutionary algorithm to solve a given task during the actual task execution.

In the second case study, we extensively studied a novel approach to increase the performance of online evolution of controllers in multirobot systems called *online hyper-evolution* (OHE). We started by studying four OHE approaches, namely **OHE-fitness+fit**, **OHE-fitness+div**, **OHE-diversity+fit**, and **OHE-diversity+div**, in which exploration and exploitation are conducted differently in the hyper level and in the base level. We then focused on improving the hyper level by combining fitness and diversity in order to devise more effective OHE procedures, and protect OHE against issues that may arise when diversity-based selection is used alone. The main conclusion of our study is that OHE represents an effective approach to online evolution of robotic controllers because it can combine the benefits of multiple algorithms over time, and can be applied to construct new algorithms during task execution.

Chapter 6

Online Behaviour Learning and Adaptation in Real Robots

Online evolution of behavioural control for robots is an open-ended approach to autonomous learning and adaptation: robots have the potential to automatically learn new tasks and to adapt to changes in environmental conditions, or to failures in sensors and/or actuators. However, researchers have almost exclusively carried out their studies in simulation because evolution in real hardware has required several days or weeks to produce capable robots (Silva et al., 2016d).

In this chapter, we focus on if and how online evolution can enable effective real-robot behaviour learning and adaptation to changing conditions in a timely manner. That is, whether or not robots can learn to solve different classes of tasks within a time-frame of one hour or less. We carry out online evolution on real Thymio II robots, and we use odNEAT to evolve neural network-based controllers. Our first research question refers to how to best seed online evolution so that controllers can be evolved in a timely manner. We assess how online evolution from an initial random population in real robotic hardware compares with online evolution seeded with controllers pre-evolved in simulations with varying degree of fidelity. We then assess the relations between simulation fidelity, task complexity, and the effort required to re-adapt online controllers that do not transfer well from simulation to reality. Although online re-adaptation of controllers was hinted at in the 1990s (Nolfi et al., 1994), it was only studied in an ad-hoc manner. Particularly, online re-adaptation was investigated without regard to how accurate a simulation needs to be in order to provide an effective bias, and in a single task (navigation and obstacle avoidance by a single robot). Our second research question is related with the unexplored adaptation and learning potential of online evolution. We therefore set up a series of adaptation experiments, in which we change the task requirements during the experiment, and we inject simulated faults in the motors of multiple robots simultaneously.

We study two single-robot tasks, specifically integrated navigation and obstacle avoidance, and homing towards a target area, and one collective robotics task, namely aggregation. The integrated navigation and obstacle avoidance task is used to analyse the performance of online evolution when two conflicting objectives have to be learned. The task implies an integrated set of actions, and thus a trade-off between avoiding obstacles and maintaining speed and forward movement. Navigation and obstacle avoidance is therefore essential for autonomous robots operating in real-world environments, and it provides the basis for more complex behaviours such as path planning (Cao et al., 1997).

The second task, homing, is an extension of navigation tasks in which a single robot starts from a random position, and must search for and navigate to a small target area located in the centre of the environment. The third task, aggregation, is a multirobot task in which dispersed robots must move close to one another so that they form and remain in a single group. Aggregation thus combines different aspects of collective robotics tasks, namely distributed search, coordinated movement, and cooperation. Furthermore, aggregation is related to a number of real-world robotics tasks, such as collective transport of heavy objects, which requires the robots to aggregate at the site of interest (Groß and Dorigo, 2009). In our aggregation task, we divide the arena into three equally-sized areas. A group of three robots must then aggregate in one of the two areas at the extremes of the arena, similarly to collective decision-making scenarios used to investigate the best-of-n decision problem in robot collectives (Valentini et al., 2015).

The outline of this chapter is the following. In Section 6.1, we describe our experimental methodology, the setup of our experiments, and the real robotic platform. In Section 6.2, we describe and discuss our experimental results. In Section 6.3, we summarise our research contribution and provide concluding remarks.

6.1 Experimental Methodology

Our experimental methodology is defined by three key components (see Figure 6.1), namely: (i) specification of the task setup, (ii) evolution in simulation, and (iii) evolution in real hardware. Each component is summarised below. Experiments were carried out in a square arena. The size of the simulated arena for evolution of the seed controllers and of the real arena was chosen to be 100 x 100 cm. The real arena was surrounded by wooden walls. Each simulated robot and each real robot was controlled by a discrete time recurrent neural network (Haykin, 1999). The inputs of the neural network were normalised sensor readings, and the outputs of the network controlled the robot's actuators. Every 100 ms, robots executed a control cycle, in which they updated sensor readings and actuation values.

The specification of the task setup was composed of:

1. Construction of the environment for the real-robot experiments, and modelling of an environment with similar characteristics in simulation.

2. Configuration of the robots' sensors and actuators, particularly the processing of raw sensory data to extract information that can be directly used by the robots' neural controllers.

3. Modelling of robot sensors and actuators in simulation. As sensors are often the main source of stochasticity in robots (Nolfi et al., 1994), we used three complementary methods to model sensory inputs: taking samples from the real robots' sensors (Miglino et al., 1995), introducing a conservative form of noise in simulated sensors (Miglino et al., 1995), and perfect readings, that is, without any stochasticity associated. For the simulated actuators, we built a differential-drive kinematics model, adjusted the model based on observations of the motor speeds of real robots, and applied a conservative amount of noise. Noise in simulated sensors and actuators corresponded to a random Gaussian component within 5% of the sensor saturation value or of the current actuation value.

4. Definition of the fitness function. In our experiments, we resorted to internal fitness functions (Floreano and Urzelai, 2000), which are based on information that is available to the robot (e.g. through its sensors

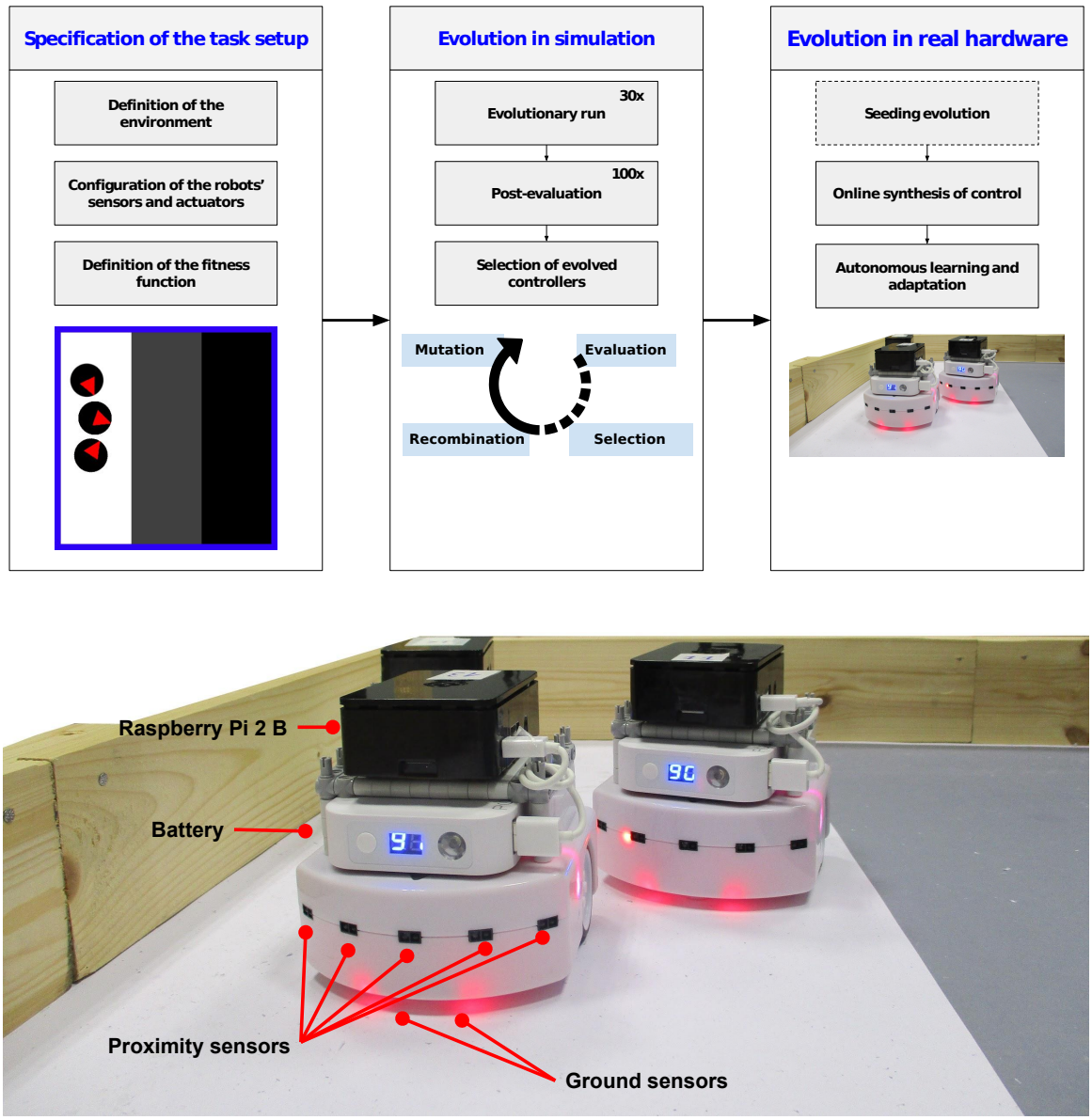


Figure 6.1: Experimental methodology (top) and real robotic platform (bottom). The experimental methodology is defined by three key components: specification of the task setup, evolution in simulation, and evolution in real hardware. Each Thymio II robot is extended with a Raspberry Pi 2 B single-board computer. The Thymio II robot has seven infrared proximity sensors (five in the front, two in the back) and two infrared ground sensors, among others.

or state of the motors). In this way, fitness assessment is self-contained, as an individual robot does not require any external devices to determine its performance.

Evolution in simulation: Our simulation-based experiments were conducted using JBotEvolver (Duarte et al., 2014), as with the experiments reported in the previous chapters. Each robot was controlled by an artificial neural network. To optimise controllers in simulation, we used the NEAT algorithm (Stanley and Miikkulainen, 2002). Each evolutionary setup consisted of 30 independent simulation-based evolutionary runs. In each run, the evolutionary process optimised a population of 100 controllers for 100 generations. A controller's performance was given by the mean fitness of 10 simulations with varying initial conditions. After each evolutionary run

ended, we carried out a post-evolution evaluation in which the top controller of each generation was evaluated in 100 simulations (without evolution) to obtain a more precise estimate of the controllers' performance. Based on the post-evolution evaluation results, we identified the highest-performing controller of each evolutionary setup.

Evolution in real hardware: To carry out our real-robot experiments, we used Thymio II robots (`https://www.thymio.org/en:thymio`). Each robot was extended with a Raspberry Pi 2 B single-board computer, see Figure 6.1. In multirobot experiments, the robots formed an IEEE 802.11g ad-hoc wireless network, and communicated with one another by broadcasting UDP datagrams.

In the experiments presented in this chapter, online evolution was either conducted from random solutions on real robotic hardware, or seeded with controllers previously evolved in simulation. For each real-robot experimental setup, five independent runs were conducted. Each run lasted one hour. Individual controller evaluations on real robots lasted 30 seconds in the navigation and obstacle avoidance task, and 25 seconds in the homing and aggregation tasks (individual evaluations in these two tasks were preceded by a five-seconds long random walk), allowing for a total of 120 evaluations per experiment.

6.1.1 Navigation and Obstacle Avoidance

In a navigation and obstacle avoidance task, a robot has to simultaneously move as straight as possible, maximise wheel speed, and avoid obstacles. The input layer of a neural network-based controller for the navigation and obstacle avoidance task has seven neurons. The input neurons receive the readings from the seven horizontal infrared sensors for obstacle detection (five in the front, two in the back), normalised to the interval [0,1]. The output layer is composed of two neurons. The values of the output neurons are linearly scaled from [0,1] to [-1,1] to set the signed speed of each wheel. The fitness score f_{nav} of a controller is given by:

$$f_{nav} = \sum_{i=1}^{T} \frac{V \cdot (1 - \sqrt{\Delta v}) \cdot (1 - d_o)}{T}, \tag{6.1}$$

where T is the length of an experiment in control cycles, $V \in [0, 1]$ is the normalised absolute sum of rotation speeds of the two wheels, $\Delta v \in [0, 1]$ is the normalised absolute value of the algebraic difference between the signed speed values of the wheels, and d_o is the highest activation value of the infrared sensors for obstacle detection. d_o takes values from 0 (no obstacle in sight) to 1 (collision with an obstacle). The three components encourage respectively, motion, straight line displacement, and obstacle avoidance.

6.1.2 Homing

Homing is an extension of navigation tasks in which a robot must move towards a point of interest in the environment. In our homing task, a single robot has to actively search for and navigate to a small target area located in the centre of the environment. The target area has a radius of 5 cm. To allow Thymio robots to sense the target area, we sampled the ground sensors' response to different colour components, and created a circular grey gradient surrounding the target area (see Figure 6.2). This configuration allows a robot to sense the target area at a distance of up to 25 cm.

The input layer of a controller for the homing task has eight neurons. The input layer receives the normalised readings of the seven horizontal infrared sensors, and the robot's proximity to the target area, scaled to a value

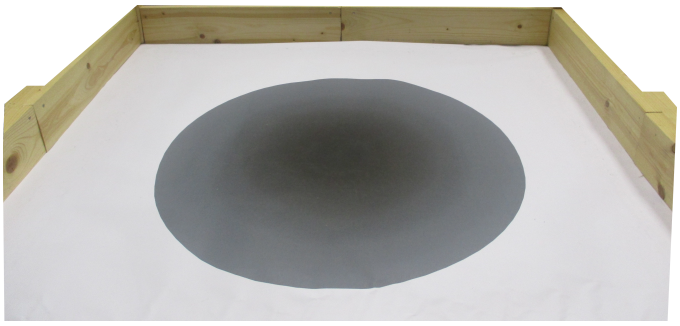


Figure 6.2: Real environment for the homing experiments.

between 0 (target area not in range) and 1 (robot is in the target area). The proximity value is computed based on the mean value of the readings from the two infrared ground sensors. Similarly to the navigation and obstacle avoidance task, the output layer contains two neurons for setting the signed speed of each wheel.

The position of the robot is randomised at the beginning of every evaluation by executing a five-seconds long random walk. Controllers are then scored based on the robot's proximity to the exterior part of the target area, up to 25 cm, for the remaining 25 seconds of the evaluation period:

$$f_{homing} = \sum_{i=1}^{T} \frac{p_t}{T}, \tag{6.2}$$

where T is the maximum number of control cycles, and p_t is the robot's proximity to the target area (linear inverse of the distance).

6.1.3 Aggregation in a Non-determined Area

In an aggregation task, dispersed robots must move close to one another so that they form a single group. In the aggregation task used in our study, we divide the arena into three equally-sized areas (see Figure 6.3). A group of three robots must aggregate in one of the two areas at the extremes of the arena, respectively denoted area A and area B.

Figure 6.3: Real environment for the aggregation and adaptation experiments.

The input layer of each controller for the aggregation task is composed of 11 neurons: seven neurons receive the normalised readings of the horizontal infrared sensors for obstacle detection; two neurons receive binary inputs indicating respectively whether the robot is in area A or not, and whether in area B or not; and two neurons receive the ratio of other robots in area A and in area B, respectively. The binary inputs are

computed based on the mean value of the readings from the two infrared ground sensors. Each robot additionally broadcasts these inputs to other robots, which allows receiving robots to compute the ratio of robots in area A and in area B. As in the previous two tasks, the output layer is composed of two neurons for setting the speed of the wheels.

Similarly to the homing task, the position of robots is randomised at the beginning of each controller evaluation. A controller is scored according to the area in which it is located and the areas in which other robots are located:

$$f_{agg} = \sum_{i=1}^{T} \frac{f_{option}}{T}, f_{option} = \begin{cases} 1 + \Delta_{area} & \text{: if robot is in area A or area B} \\ 0 & \text{: otherwise} \end{cases} \tag{6.3}$$

where T is the duration of the experiment in number of control cycles, and Δ_{area} is the number of other robots in the same area.

6.1.4 Adaptation Experiments

Task adaptation experiments: In these experiments, we first evolve controllers to solve the aggregation task, in which robots must aggregate either in area A or in area B, and we then change the task so that robots have to aggregate in the centre of the arena (see Figure 6.3). A controller is scored based on:

$$f_{agg\_centre} = \sum_{i=1}^{T} \frac{f_{centre}}{T}, f_{centre} = \begin{cases} 1 + \Delta_{centre} & \text{: if robot is in centre area} \\ 0 & \text{: otherwise} \end{cases} \tag{6.4}$$

where T is the duration of the experiment in number of control cycles, and Δ_{centre} is the number of other robots in the centre area. While intuitively simple, this change in the task requirements forces robots to radically modify their response to key sensory inputs.

Fault injection experiments: We inject simulated faults in the motors of real robots evolved to solve the aggregation task. We conduct three sets of experiments, in which we respectively inject faults in the motors of $N = \{1, 2, 3\}$ robots. The fault injection procedure consists of: (i) selecting the required number of robots, (ii) randomly choosing either the left or the right wheel of each of the N robots, and (iii) randomly limiting the wheel speed to 50%–75% of its maximum value. Such faults are analogous to failures in servo drives and motor-amps of mobile robots operating for long periods of time (Carlson et al., 2004).

In both the task adaptation and the fault injection experiments, we resume the evolutionary process after changes take place. Specifically, robots first evolve for one hour to solve the initial version of the aggregation task, as described in the previous section. We then select the group of real robots with highest fitness on average, and allow the robots to evolve for another hour in order to adapt to changes.

6.1.5 Performance Analysis and Treatments

We have empirically determined performance-based criteria to distinguish between successful controller evaluations and failed controller evaluations. In the integrated navigation and obstacle avoidance task, a controller evaluation is considered to be successful if the fitness score f_{nav} of the controller is above 0.5 (e.g. the robot effectively avoids obstacles and navigates at least at half of the maximum speed, or moves at maximum speed and can avoid obstacles before they are at a distance of half of the sensor range). In the homing task, a controller

evaluation is considered successful if the robot is able to find the target area, and stay in it for at least three consecutive seconds. Similarly, an evaluation in the aggregation task is considered successful if all robots in the group can coordinate and remain in the same area for at least three consecutive seconds. In the homing and in the aggregation task, the controller success vs. failure analysis is respectively complemented by the amount of time that a robot has spent in the target area, and by the amount of time that a group of robots has spent aggregated.

We compare results along three dimensions: (i) fitness, that is, the performance or quality of a controller, (ii) number of successful controller evaluations in a given run and/or experimental setup, and (iii) behaviour distance, that is, how distinct the actions performed during task execution by one or more reference controllers are with respect to the actions performed by other controllers. To compare behaviours, we resorted to a generic Hamming distance-based behavioural metric based on the mapping between sensors and actuators, see Section 4.1.1. To understand how the evolutionary process proceeds, we construct two-dimensional *behaviour-fitness maps*, which relate behaviour distance with task performance.

We use the two-tailed Mann-Whitney test to compute significance of differences between sets of results because it is a non-parametric test, and therefore no strong assumptions need to be made about the underlying distributions. Success rates are compared using the two-tailed Fisher's exact test, a suitable non-parametric test (Fisher, 1925). When multiple comparisons are performed, we adjust the ρ-value using the two-stage Hommel method (Hommel, 1988).

6.2 Experimental Results

The presentation of experimental results in structured as follows. In Section 6.2.1, we describe the evolutionary synthesis of controllers in simulation, and how the highest-performing controllers transfer to real robotic hardware. In Section 6.2.2, we report on how to best seed online evolution in real hardware, that is, how online evolution from an initial random population compares with online evolution seeded with controllers pre-evolved in simulation. In Section 6.2.3, we describe if and how online evolution can enable adaptation to unforeseen circumstances, namely changes in task requirements and faults injected in the motors of robots. In Section 6.2.4, we detail and discuss how the properties and different algorithmic components of odNEAT influence timely controller synthesis.

6.2.1 Transferring Simulation-evolved Controllers to Real Robots

In this section, we assess how the highest-performing controllers found in simulation transfer to real robots. First, we evolved controllers in simulations with varying degree of fidelity by using three complementary methods to model sensory inputs: taking samples from the real robots' sensors (Miglino et al., 1995), introducing a conservative form of noise in simulated sensors (Miglino et al., 1995), and perfect readings, that is, without any stochasticity associated. Using samples from real sensors increases the accuracy of simulations because a more realistic sensor model is employed (Miglino et al., 1995), which in turn has the potential to reduce the difference between the sensory input experienced in simulation and in reality. Conservative noise, on the other hand, has been widely adopted (Silva et al., 2016d) because it has the potential to promote the evolution of robust controllers that can tolerate variations in the sensory inputs profile, including differences between inputs

Table 6.1: Controller performance in simulation and in reality. In the navigation and obstacle avoidance task and in the homing task, the fitness scores can take values in the interval [0,1]. In the aggregation task, the fitness scores can take values in the interval [0,3].

Task	Variants	Simulation	Real robot (mean)	Real robot ([worst, best])
Navigation	No noise	0.81	0.57	[0.52, 0.59]
	Conservative noise	0.83	0.01	[0.00, 0.04]
	Real samples	0.85	0.81	[0,79, 0.83]
Homing	No noise	0.72	0.02	[0.00, 0.04]
	Conservative noise	0.72	0.02	[0.00, 0.06]
	Real samples	0.74	0.42	[0.17, 0.62]
Aggregation	No noise	2.69	0.80	[0.37,1.44]
	Conservative noise	2.69	0.81	[0.36,1.38]
	Real samples	2.69	0.81	[0.32,1.43]

in simulation and in real robots.

The fitness trajectories throughout evolution and the post-evolution evaluation fitness scores are shown in Figure 6.4. Solutions for the three tasks were found within 100 generations for all sensor modelling methods. In the simulation-based post-evolution evaluation, controllers evolved using the real samples method significantly outperform those evolved using the no noise and conservative noise modelling methods in the navigation and obstacle avoidance task and in the homing task ($\rho < 0.0001$, Mann-Whitney). In the aggregation task, differences in the post-evolution evaluation fitness scores are not significant ($\rho > 0.05$).

In real robots, we assessed the highest-performing controller found for each task in simulation, and we compared the performance levels of controllers in simulation and in real robotic hardware. Table 6.1 shows how the best controller found in simulation for a given experimental configuration transfers to real robotic hardware (five real-robot evaluations per controller). In the navigation and obstacle avoidance task, the real samples method is the one that enables the most effective transfer, as controllers yielded similar performance levels in simulation and in reality. Controllers evolved with the conservative noise approach failed to transfer successfully to real robots. Controllers evolved without noise in the simulated sensors transferred from simulation to reality with average fitness decreases of 0.24. In the homing task, the real samples method was also the most effective one, although significant decreases in performance were typically observed. Controllers evolved with noise and controllers evolved without noise failed when executing on the real robots. Visual inspection of the behaviour revealed that controllers that failed in the navigation task and in the homing task typically could not resume operation after being in close proximity and/or colliding with the walls of the arena. For example, in the homing task, this circumstance caused the failure of evolved behavioural strategies in which the robot moved forward until a wall was in close proximity, and then backwards towards the target area. That is, controllers were not prepared for specific sensory conditions that they encountered in the real environments, conditions that controllers evolved using samples from the real robots are less prone to suffer from. In the aggregation task, the transfer of controllers from simulation to reality consistently resulted in a significant performance degradation.

The results presented above indicate that more accurate simulations, such as those that make use of samples from real robots, typically facilitate better transfer of controllers from simulation to reality. The conservative noise approach, although widely adopted (Silva et al., 2016d), is sensitive to the experimental conditions as it cannot guarantee that the relevant sensorimotor conditions are present in simulation (e.g. specific view of the

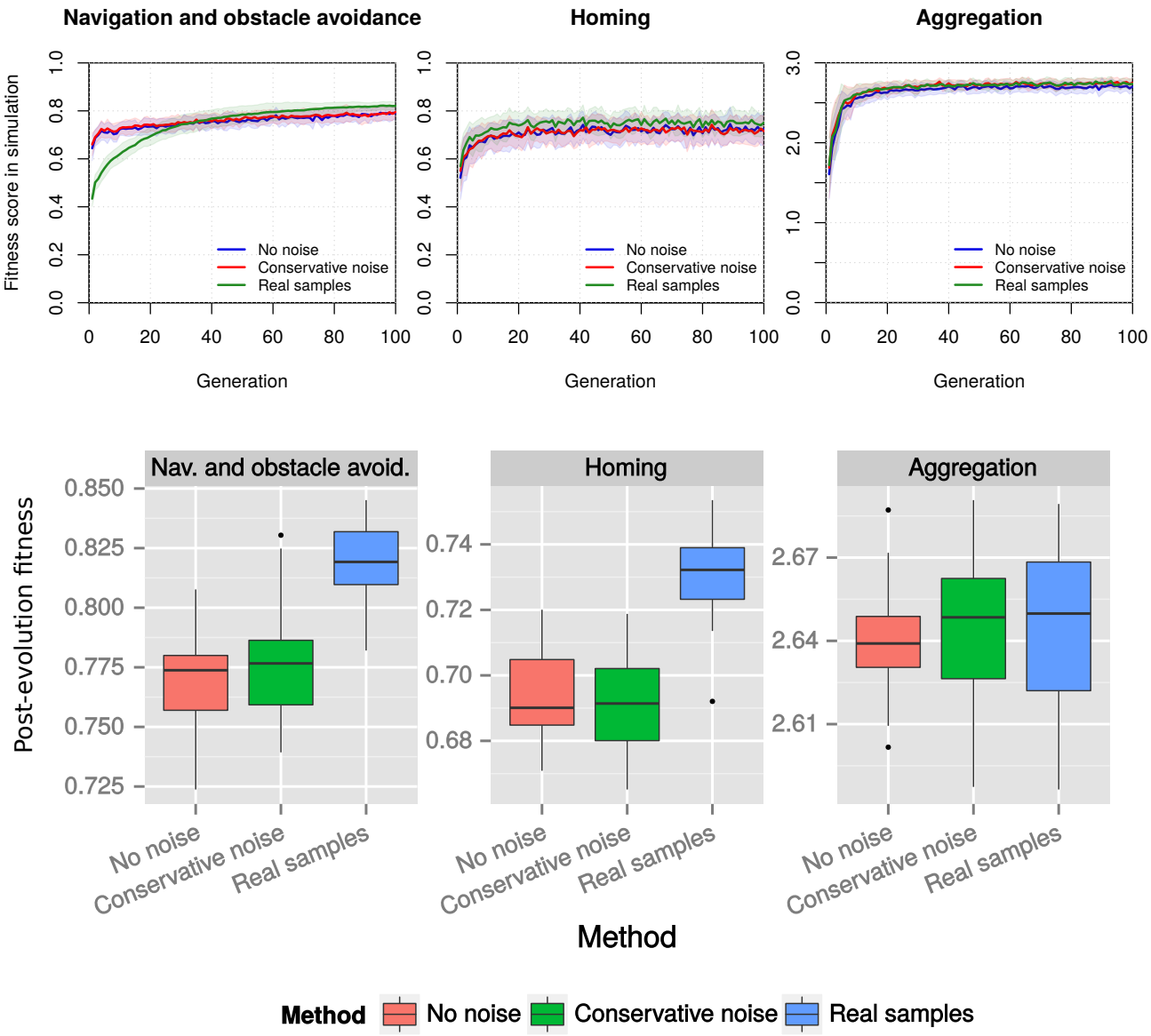


Figure 6.4: Fitness plots for the simulation-based experiments. Top: highest fitness scores found so far at each generation, for a given sensor modelling approach. The lines depict the mean of 30 runs, with the respective standard deviation shown in lighter colours. Bottom: distribution of post-evolution evaluation fitness scores for the highest-performing controller of every simulation-based run. The lower limit and the upper limit of boxplots are respectively: 0.72 (no noise) and 0.85 (real samples) in the navigation and obstacle avoidance task, 0.67 (no noise and conservative noise) and 0.75 (real samples) in the homing task, and 2.59 and 2.69 (conservative noise and real samples) in the aggregation task.

walls of the arena from the robot's vantage point). Nonetheless, similar issues affect the real samples method as the task becomes more challenging (e.g. when multiple robots are involved). As more types of objects are considered or richer robot-environment interactions exist in the environments, the more samples have to be taken from a large number of positions because there are multiple unique readings of the objects depending on the robot's relative location and orientation (Silva et al., 2016d). If such sampling is not performed, it introduces the potential for robots to exploit poorly modelled phenomena, resulting in sub-optimal performance

when controllers are deployed to real robots.

6.2.2 Evolution of Control in Real Robots

In this section, we evaluate how online evolution from an initial random population in real hardware, henceforth called *base evolution*, compares with online evolution seeded with controllers pre-evolved in simulation. We address three key research questions: (i) to what extent the online evolutionary process can be accelerated by being seeded with controllers pre-evolved in simulation, (ii) how accurate a simulation needs to be in order to provide an effective bias, and (iii) how much evolutionary effort is required to adapt pre-evolved controllers that do not transfer well from simulation to real robotic hardware. The highest-performing controller of each simulation-based setup (real samples, conservative noise, no noise) is subsequently used in a set of real-robot experiments, in which it is specified as the initial controller of odNEAT.

Influence of the Seeding Process on Controller Evolution and Re-adaptation

In the navigation and obstacle avoidance task, the mean percentage of successful controller evaluations per robot is 38.0% ± 16.1% for the base evolution setup, 9.5% ± 9.0% for the no noise setup, 6.5% ± 12.3% for the conservative noise setup, and 85.8% ± 3.6% for the real samples setup. The number of successful controllers evolved with the real samples method is significantly superior to the number of successful controllers evolved with other methods ($\rho < 0.0001$, Fisher's exact test). The base evolution method also outperforms the no noise method and the conservative noise method ($\rho < 0.0001$).

Figure 6.5 shows two-dimensional behaviour-fitness maps, which relate behaviour distance with task performance, for the four setups of the navigation and obstacle avoidance task. To elucidate on how evolution proceeds in the behaviour space, behaviour distance is relative to the initial controllers. The maps show that the progress of the online evolutionary process through the behaviour space is strongly influenced by the choice of seed controllers. In the base evolution setup, the evolutionary process mainly discovers a certain class of behaviours (those with behaviour distance between 80 to 100, left part of the map). In the no noise setup, evolution synthesises behaviours at varying distances from the seed controllers, but can rarely reach high-fitness regions. This result indicates that seeding evolution with controllers from the no noise setup biases the search towards uninteresting regions of the behaviour space. In these regions, high-performing controllers are challenging to synthesise, hence the comparatively low rate of successful evaluations.

If evolution is seeded with controllers from the conservative noise setup, it performs similarly to when it is seeded with controllers from the no noise setup. However, the evolutionary process performs a more localised search for solutions in behaviour space (behaviour distances between ~50 and ~100). This result indicates that the bias induced by seed controllers is stronger, which in turn increases the re-adaptation effort required to evolve effective controllers. The final map shows that if evolution is seeded with controllers from the real samples setup, it is able to consistently synthesise high-performing solutions to the task (top-left region of the map). The main result is that a localised search around the controllers considered as effective in simulation is sufficient to achieve high-performing controllers in real robotic hardware.

The results presented in this section indicate that simulations with different sensor models impose different constraints on the online evolutionary and re-adaptation process. In the following sections, we assess if and

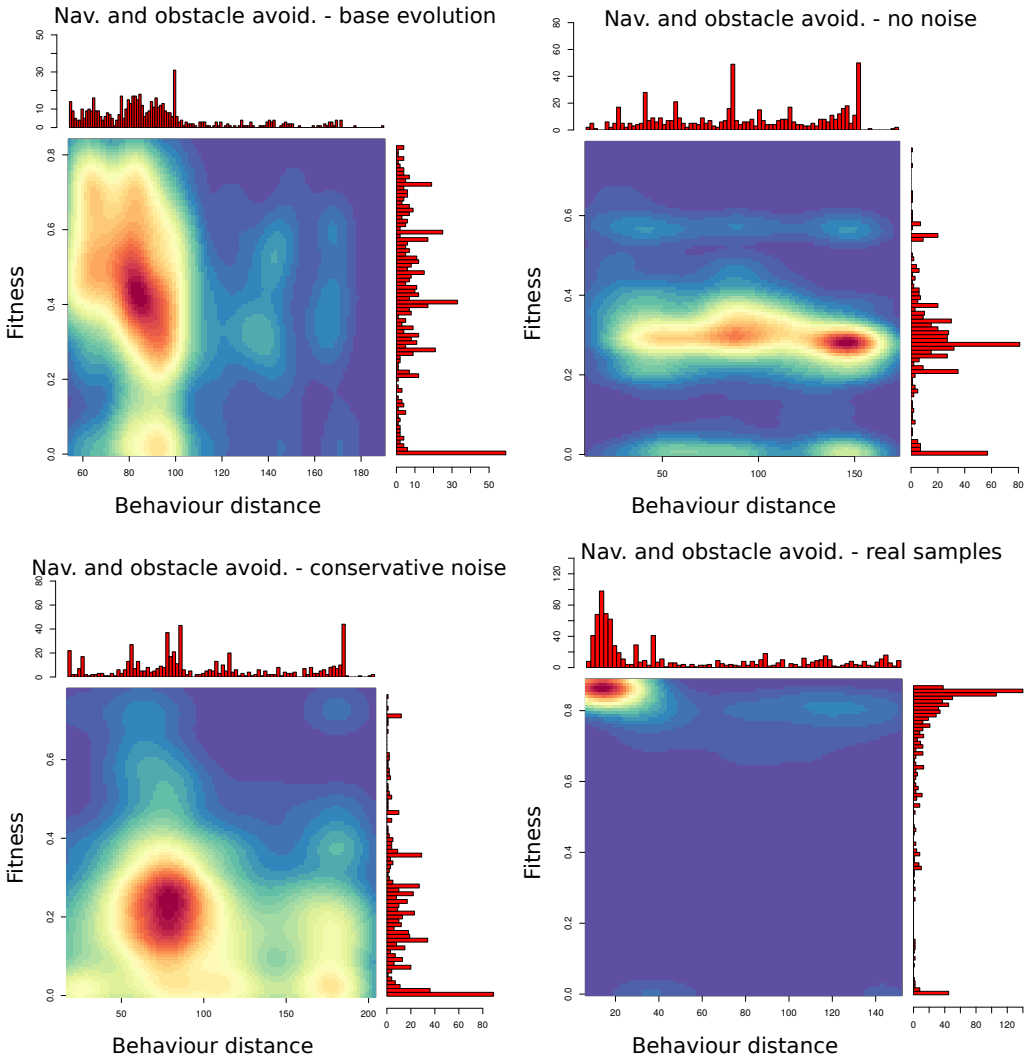


Figure 6.5: Behaviour-fitness map for the navigation and obstacle avoidance tasks. Behaviour distance is relative to the initial controllers: a randomly generated controller for the base evolution setup (top left), and the seed controller for the no noise setup (top right), conservative noise setup (bottom left), and real samples setup (bottom right). We empirically determined that fitness scores higher than 0.9 cannot be obtained in our enclosed arena.

how the results generalise to more challenging tasks (e.g that require searching for a target area or group coordination), specifically the homing task and the aggregation task.

Challenging the Reality Gap with Online Evolution

In the homing task, the mean percentage of successful evaluations per robot is 3.7% ± 0.7% for the base evolution setup, 6.3% ± 2.0% for the no noise setup, 3.8% ± 2.9% for the conservative noise setup, and 4.2% ± 1.7% for the real samples setup. Differences are not statistically significant across any comparison ($\rho > 0.05$, Fisher's exact test).

Figure 6.6 summarises the amount of time spent in the target area by the controllers that can successfully carry out the homing task. The median amount of time in the target area is 12 out of 25 seconds in the base evolution setup, and 9 out of 25 seconds for other setups. These combined results furthermore show that:

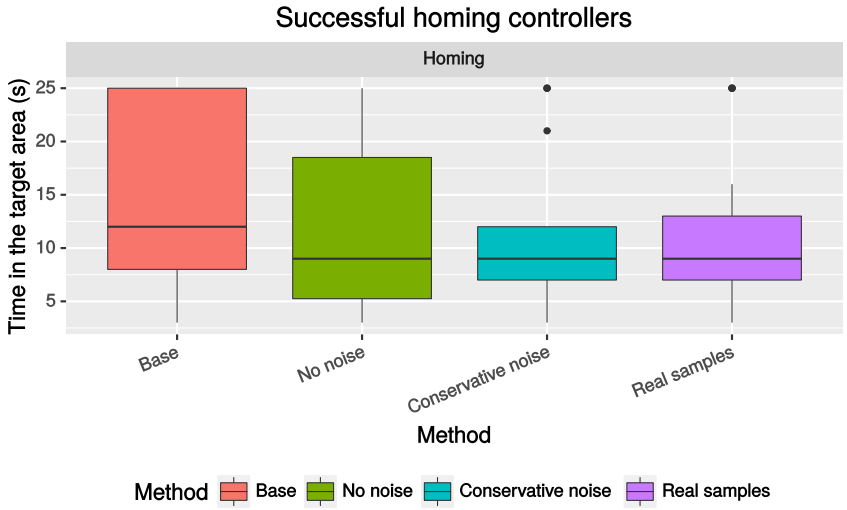


Figure 6.6: Successful controllers for the homing task. Time spent in the target area by the controllers that can successfully carry out the homing task. The lower limit and the upper limit of boxplots is respectively 3.00 (all methods), and 25.00 (base and no noise).

(i) effective solutions to the task can be evolved from random solutions on real robots in a timely manner, and that (ii) online evolution is able to re-adapt controllers that do not transfer well from simulation to real robotic hardware. Importantly, the fact such re-adaptation actually occurs in one hour or less, which is the time-frame considered in our experiments, challenges the view that completing the optimisation process in a real robot is only meaningful if solutions found in simulation are close to solutions in reality, that is, if the reality gap is sufficiently narrow (Koos et al., 2013).

Evolution of Collective Behaviours

In the aggregation task, the mean percentage of successful evaluations per group of robots is 11.5% ± 4.0% for base evolution, 14.8% ± 11.2% for the no noise setup, 8.7% ± 5.3% for the conservative noise setup, and 9.0% ± 3.2% for the real samples setup. Differences are not statistically significant across any comparison ($\rho > 0.05$, Fisher's exact test).

Figure 6.7 summarises the amount of time that groups of robots able to carry out the aggregation task have spent in the same area. The median amount of time aggregated is approximately 8 out of 25 seconds for all experimental setups. The highest-performing groups of controllers, that is, those that aggregate the longest, are typically synthesised either in the base evolution setup or in the real samples setup. These results demonstrate that online evolution allows robots to effectively evolve solutions to the aggregation task. To the best of our knowledge, this experiment is the first instance of online evolution of collective and coordinated behaviours in a timely manner. Previous studies involving multiple robots have been limited to individual tasks such as phototaxis (Watson et al., 2002), dynamic phototaxis (Bredeche et al., 2012), and foraging (Heinerman et al., 2016), and have required significantly longer evolution times (Watson et al., 2002; Bredeche et al., 2012).

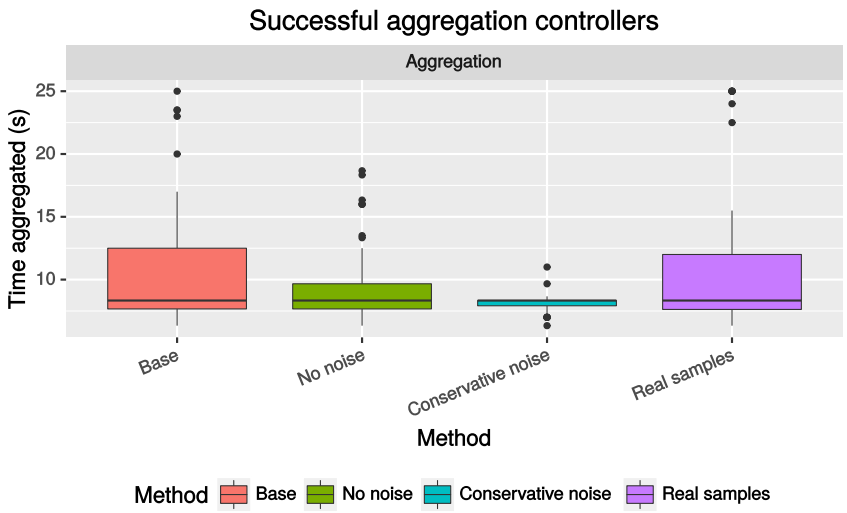


Figure 6.7: Successful controllers for the aggregation task. Amount of time that successful aggregation controllers spent in the same area. The lower limit and the upper limit of boxplots is respectively 6.33 (base, no noise, and real samples), and 17.00 (base).

6.2.3 Behaviour Adaptation and Fault Tolerance in Real Robots

In this section, we assess if and how robots can adapt and learn new behaviours when task requirements change and when faults are injected in the motors, see Section 6.1.4 for details on the experimental protocol. Using the aggregation task, we devised two experimental setups to assess the ability of online evolution to adapt behavioural control, namely: (i) *task adaptation experiments*, in which we change the configuration of the aggregation task so that robots have to aggregate in the centre of the arena after learning to successfully aggregate in one of the extremes, which forces robots to interpret their sensory information in a radically different way, and (ii) *fault-injection experiments*, in which simulated faults analogous to failures in servo drives and motor-amps (Carlson et al., 2004) are injected in the motors of $N = \{1, 2, 3\}$ out of 3 robots after they have learned to solve the initial version of the aggregation task. The different configurations of the fault-injection experiments are respectively called the *one-fault setup*, the *two-faults setup*, and the *three-faults setup*.

Task Adaptation Results

The mean percentage of successful evaluations per group of robots is 11.7% ± 8.8% for the task adaptation experiments, which is comparable to that of the base evolution experiments, despite a higher standard deviation: 11.5% ± 4.0% of successful evaluations. Differences in successful controller evaluations are not statistically significant ($\rho > 0.05$, Fisher's exact test).

The analysis of the amount of time that robots have spent aggregated in the different adaptation experiments (see Figure 6.8) indicates that robots are able to effectively adapt to the new circumstances. In the task adaptation experiments, robots reach performance levels similar to those of base evolution: they learn to effectively aggregate in the centre area and thus to solve a new instance of the aggregation task in a timely manner. Importantly, the evolutionary process focuses mainly on optimising a certain class of behaviours in order to re-adapt controllers to the new task (see Figure 6.9 for details).

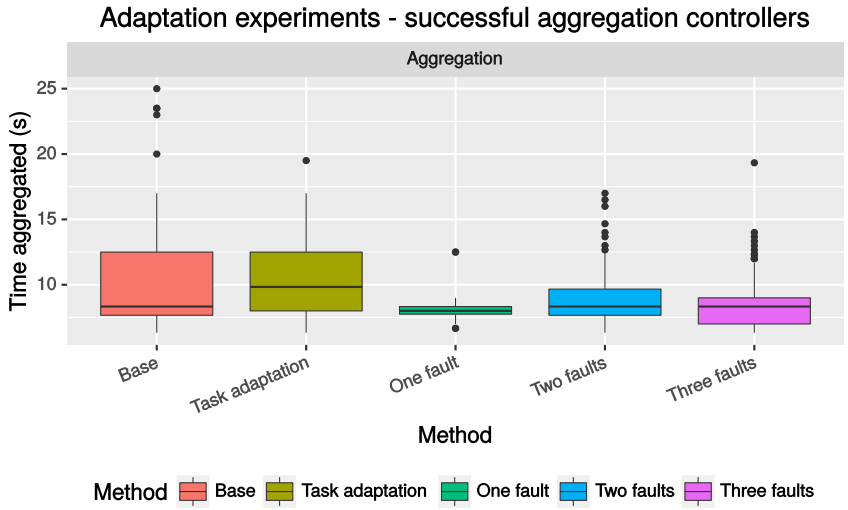


Figure 6.8: Successful controllers for the adaptation experiments. Amount of time that controllers spent in the same area. The lower limit and the upper limit of boxplots is respectively 6.33 (base, task adaptation, two faults, and three faults), and 17.00 (base and task adaptation).

Fault Injection Results

The mean percentage of successful evaluations per group of robots is 3.0% ± 1.3% for the one-fault setup, 11.3% ± 9.9% for the two-faults setup, and 14.7% ± 7.9% for the three-faults setup. Interestingly, the fault injection progressively leads to more successful evaluations as the number of faults increases. Successful controller evaluations are significantly higher in the base evolution setup, two-fault setup, and three-fault setup than in the one-fault setup ($\rho < 0.0001$, Fisher's exact test).

In the fault-injection experiments, successful controllers cause robots to aggregate for a shorter amount of time when compared with the base evolution setup (see Figure 6.8). The reason is that faulty robots, even after they learn to cope with a faulty wheel, are still limited in the speed they can achieve (from 50% to 75% of the maximum speed). As a result, robots require more time to find one another and aggregate. In the one-fault setup, only one robot is in a faulty state. The faulty robot has to learn to overcome the fault by itself, as controllers evolved by the other two robots in the group are optimised for non-faulty wheels, and cause the faulty robot to move in circles. In this way, the one-fault setup yields a comparatively lower percentage of successful controller evaluations per group of robots.

In the two-faults setup, the stochasticity of the fault injection causes the evolutionary process to commonly optimise behavioural control for a group of robots in which there is: (i) one non-faulty robot, (ii) one robot with a simulated fault in its left wheel, and (iii) one robot with a simulated fault in its right wheel. Consequently, evolution focuses on behaviours that differ more from the initial controllers than those evolved in the one-fault setup (see Figure 6.9 for the behaviour-fitness map). The increased behaviour exploration is beneficial, as the performance levels of robots increase with respect to the one-fault setup both in terms of the successful controller evaluations, and of the amount of time that the group of robots was aggregated. The results of the three-faults setup are similar to those of the two-faults setup. Although the amount of time aggregated is comparable, both the percentage of successful controller evaluations and the behaviour distance to initial controllers are higher in

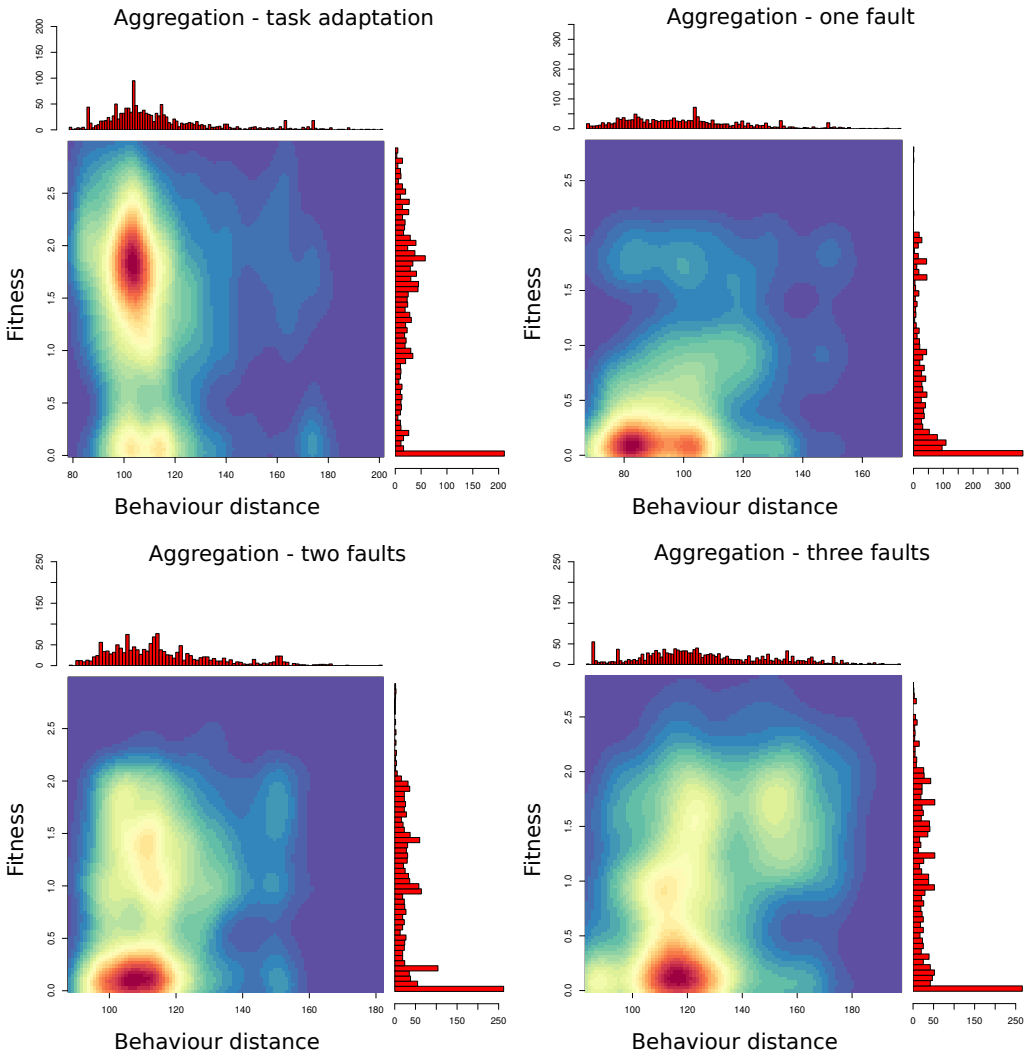


Figure 6.9: Behaviour-fitness map for the adaptation experiments. Behaviour distance is relative to the controllers that were executing when: the task requirements changed (top left), one fault was injected (top right), two faults were injected (bottom left), and three faults were injected (bottom right).

the three-faults setup than in the two-faults setup.

6.2.4 Ablation Studies

One key component in our experimental methodology is the odNEAT algorithm. Using the aggregation task, we conducted four sets of ablation experiments in which odNEAT executed: (i) without the genotypic diversity mechanisms (*no niching*), (ii) without the crossover operator (*no crossover*), (iii) with *no exchange* of genomes between robots, and (iv) with a *minimal population* of size 1, which combines ablations (i), (ii), and (iii) because the internal population of each robot can only hold the genome corresponding to the controller that is executing. The minimal population ablation transforms odNEAT in a variation of the (1+1)-online algorithm (Bredeche et al., 2009), which was proposed after the classic (1+1) evolutionary strategy, and upon which multiple online evolution algorithms are based (Bredeche et al., 2012; Silva et al., 2015e).

The mean percentage of successful evaluations per group of robots is 11.5% ± 4.0% for base evolution

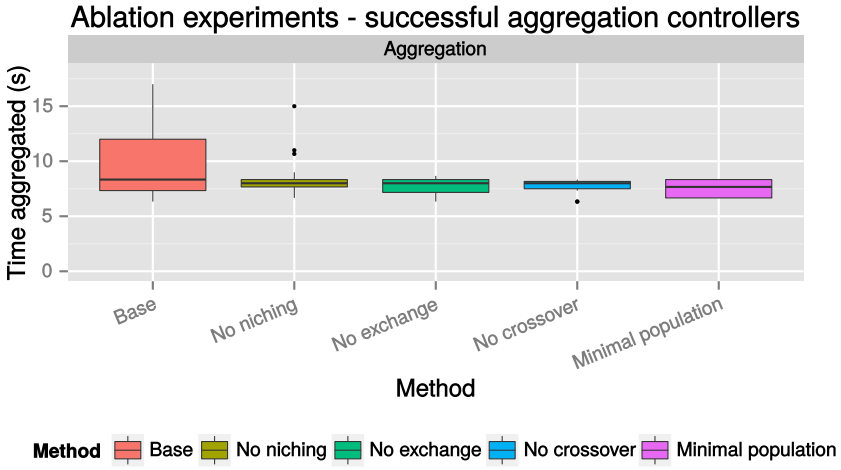


Figure 6.10: Successful controllers for the ablation experiments. Amount of time that robots have spent aggregated. The *Base* evolution setup yields five outliers above 18s, namely 20s, 23s, 23.5s, 23.5s, and 25s (not shown for better reading of the plot). The lower limit and the upper limit of boxplots is respectively 6.33 (base, and no exchange), and 17.00 (base).

(non-ablated odNEAT), 5.7% ± 1.7% when the niching scheme is ablated, 3.2% ± 1.8% when the crossover operator is ablated, 2.5% ± 1.0% when the exchange of genomes is ablated, and 1.5% ± 1.5% when the ablations are combined (minimal population ablation). The non-ablated version of odNEAT enables successful controller evaluations significantly more often than every ablated version ($\rho < 0.05$, Fisher's exact test). Complementarily, Figure 6.10 shows that the non-ablated version of odNEAT generates more effective controllers, which are able to form and remain in a single group for a larger amount of time. The main conclusion of these combined results is thus that each component of odNEAT contributes to the algorithm's performance as an efficient online evolution algorithm. In effect, the results of the real-robot ablation studies are consistent with the results of the simulation-based ablations studies (see Section 3.5.2), which have also demonstrated the importance of the internal population, of the exchange of genomes between robots, and of the crossover operator.

6.3 Summary

In this chapter, we reported successful evolution on neural network-based controllers in real robotic hardware to solve two single-robot tasks and one collective robotics task. Controllers were evolved either from random solutions or from solutions pre-evolved in simulation. In all cases, capable solutions were found in a timely manner (one hour or less). We furthermore demonstrated for the first time the adaptive capabilities of online evolution in real robotic hardware, including a robot collective able to overcome faults injected in the motors of multiple robots simultaneously. We concluded by showing that one key enabler of the robots' ability to effectively learn and adapt is the underlying evolutionary algorithm, odNEAT.

Chapter 7

Conclusions and Future Work

This research has been inspired by our longstanding vision that robots should not need to be programmed for a particular task. Instead, we see roboticists of the future only as providers of a task specification to robots; and robots as smart, reliable, mobile machines that autonomously learn to solve the task, and that can adapt their behaviour to changing environments and task conditions without human intervention. However, at the current state of the development, there are different issues precluding progress towards our vision, namely (Brooks and Matarić, 1993; Silva et al., 2016d): (i) the behavioural control system of robots is typically based on predefined rules that remain fixed during task execution, and thus robots tend to have limited learning abilities, and (ii) behaviour learning approaches typically require several hours or days to produce capable robots.

In this thesis, we studied new ways to enable efficient online learning in groups of autonomous robots through an open-ended approach called online evolution. Our main research contributions can be distilled into the following:

- We reviewed and analysed the state of the art in evolutionary robotics, including the open issues in the field (Silva et al., 2016a,d).
- We extensively assessed an approach called odNEAT (Silva et al., 2015e) for online evolution of artificial neural network-based controllers in multirobot systems.
- We studied the dynamics of online evolution of controllers at two different scales (Silva et al., 2013, 2015b), namely the dynamics of distinct neuronal models (microscopic scale), and the scalability properties of online evolution with respect to group size (macroscopic scale).
- We developed two novel approaches to accelerate online evolution of controllers in multirobot systems (Silva et al., 2016b), namely a racing approach that allows individual robots to cut short the evaluation of poor controllers, and a population cloning approach that enables each robot to clone and transmit a varying number of high-performing controllers stored in its internal population to other robots nearby.
- We proposed and assessed online hyper-evolution (Silva et al., 2016c, 2017c), an approach that allows robots to improve their learning process as they operate in the task environment by changing to a better suited algorithm, or by constructing novel algorithms during task execution.
- We carried out online evolution of neural network-based controllers in real robotic hardware. Controllers were evolved to solve two single-robot tasks and one collective tasks in a timely manner (one hour or

less). We furthermore demonstrated for the first time the adaptive capabilities of online evolution in real robotic hardware, including a robot collective able to overcome faults injected in the motors of multiple robots simultaneously (Silva et al., 2017a,b).

In summary, our experiments and results have shown that online evolution has the potential to become a viable approach to the synthesis of control for real robotic systems. This research has contributed to bringing online evolution closer to widespread adoption at multiple levels. We have developed a number of novel algorithms and approaches to increase the performance of online evolution of robotic controllers, introduced unprecedented methodologies such as online hyper-evolution, and conducted systematic real-robot experiments to validate the ability of online evolution to both create and adapt the behavioural control of robots in a timely manner. Below, we discuss challenges and avenues of research with respect to scale online evolution to real-world applications, and to raise the level of generality at which online evolution approaches can operate.

7.1 Future Work

7.1.1 Self-supervised Behaviour Learning

One of the limitations of online evolution is the controller evaluation policy. In the real-robot experiments described in Chapter 6, we employed an evaluation policy in which the evolutionary process takes place at regular intervals: each controller is assessed for a fixed, predefined amount of time, after which a new controller is generated. This policy is employed in the vast majority of online evolution algorithms. However, in real-world tasks, continuously producing new controllers can potentially harm individual task execution, and disrupt group behaviours (e.g. one robot changes from a high-performing controller to a low-performing controller when a robot collective is pushing an object towards a target area) (Silva et al., 2015e). Alternatively, policies with dynamic evaluation periods could be employed, as in the original definition of odNEAT, see Chapter 3, and/or the experiments reported in Chapter 4 and in Chapter 5. In this context, a new controller is only synthesised if the current one is considered to have failed (e.g. carried out a given number of low-performing actions), that is, when it is actually necessary. However, such approaches are ultimately designed by the experimenter and may be too conservative, thus delaying the synthesis of high-performing solutions (Silva et al., 2015b).

For online evolution to scale to real-world applications, we argue that more sophisticated and task-independent techniques to *control* the evolutionary process and determine *when* robots should learn new behaviours are necessary. A potential solution is enabling robots to autonomously enter new adaptation periods by *self-supervising* their behaviour learning process. One field of research from which inspiration can be sought is that of *artificial curiosity* (Oudeyer et al., 2007), in which agents have an internal drive that pushes them towards situations in which, for example, reward-based learning progress is maximised. Originally introduced in the reinforcement learning domain, the artificial curiosity framework has been used for active learning tasks (Schaul et al., 2011).

7.1.2 Improving Online Hyper-Evolution

In Chapter 5, we extensively assessed online hyper-evolution (OHE), an unprecedented approach in evolutionary robotics. The ability of OHE to construct algorithms opens a new path in online evolution of behavioural control because it allows robots to improve their own learning process. There are, however, two potential limitations

in OHE. First, OHE does not explicitly attempt to achieve scalability with respect to the number of possible configurations in the search space. If the number of algorithmic components is in the order of hundreds or thousands, then the search space is likely to become prohibitively large for the current implementation of OHE. As such, one possible extension of OHE is the addition of intelligent techniques that could probabilistically infer the potential relationship between similar components (e.g. Bayesian optimisation with Gaussian process priors for prediction). Another limitation of OHE is that, while the approach can handle discrete values (e.g. select algorithms or components), it currently cannot optimise continuous parameters such as mutation and/or crossover rates, among others. In future work, we plan to address these issues and extend OHE to enable online optimisation of continuous learning parameters.

7.1.3 Scaling the Evolutionary Process to More Complex Tasks

One concern in evolutionary robotics, and thus in online evolution, is scaling to tasks that require higher behavioural complexity (Silva et al., 2016d). For general-purpose robots capable of learning a variety of different skills for different tasks to become feasible, it is required that they retain old skills while learning new skills. However, such objective remains elusive because robots and artificial agents tend to forget previously learned information in a *catastrophic manner* (Ellefsen et al., 2015) when multiple skills are required. For example, in the optimisation of artificial neural networks, old skills may be lost because the weights that encode them are modified in order to enable learning new tasks.

One solution to allow robots to *accumulate* skills throughout their lifetime is to rely on modular control systems in which different skills are represented as *building blocks* of the system, and then combined in order to solve more complex tasks. With the macro-neurons approach (Silva et al., 2014a), we have taken the first steps towards this goal by enabling building blocks to be evolved or preprogrammed, seamlessly specified in the neural structure, and then optimised together with the neural networks' weights and topology in a unified manner. Future work should explore the usage of this and similar techniques to enable accumulation of learned behaviours, as well as techniques to effectively evolve complex individual skills (Silva et al., 2015d).

7.1.4 Recovering from Faults and Damage via Evolution of Robot Body Plans

The use of robotic systems in real-world conditions often implies hardware malfunctions, faults, and damages caused by prolonged use and environmental factors. Our real-robot experiments have shown that robots can optimise their control system in order to adapt to cope with faults injected in the motors. However, in real-world applications, it may be necessary to expand the space of evolutionary optimisation to include online adaptation of how morphological features, sensors, and actuators are used online. For example, when a robot suffers an injury, the standard damage recovery in robotics typically involves two phases: self-diagnosis, and selecting a contingency plan or compensatory behaviour provided by the experimenter (Cully et al., 2015). Because robot engineers are not able to anticipate every possible situation beforehand, being able to adapt the robot's morphology by altering the body plan becomes an important aspect (Bongard et al., 2006). Comparing with more traditional approaches, one of the main strengths of online evolution is that it is independent of control systems or types of robots, such as four-legged and eight-legged robots. In this way, online evolution yields significant potential to enable autonomous recovery from faults and damage.

Appendix A

The Role of the Genomic Encoding in the Evolution of Complex Controllers

As discussed by Husbands et al. (1997), a central aspect in neurovolution of controllers is the *genomic encoding*. A key commonality among the majority of past and current studies is the use of a *direct encoding* (Silva et al., 2016d), in which each parameter of an artificial neural network (ANN) is independently specified and optimised. As a result, direct encoding methods tend not to perform well in high-dimensional and in epistatic tasks, and are therefore unsuited for the evolution of robot behaviours that require complex large-scale ANNs (Husbands et al., 1997; Meyer et al., 1998).

A.1 Evolving Complex Controllers with Indirect Encodings

Indirect encodings, also called generative or developmental encodings, enable representational efficiency in evolutionary algorithms by incorporating concepts from evolutionary developmental biology. The indirect encoding process is inspired by the genetic reuse that allows for structures to be represented compactly in DNA (Stanley and Miikkulainen, 2003). For example, while there are 20,000 to 25,000 human protein-coding genes (Southan, 2004), adult humans are composed of roughly 100 trillions of cells (Stix, 2006), and the cerebral cortex alone has approximately 19×10^9 neurons (Herculano-Houzel, 2009) and 6.7×10^{13} synapses (Murre and Sturdy, 1995). In contrast with direct encodings, indirect encodings are more compact, as the same gene can be reused multiple times to construct different parts of the phenotype. That is, indirect encodings allow solutions to be represented as *patterns* of parameters, rather than requiring each parameter to be represented individually (Bentley and Kumar, 1999; Stanley and Miikkulainen, 2003; Bongard, 2002; Seys and Beer, 2007; Risi, 2012). In this way, evolution can search in a low-dimensional space and generate arbitrarily larger controllers.

Development is a prominent feature of biological organisms that enables the large-scale nervous systems underlying intelligence. Outside evolutionary robotics, several researchers have studied how to apply developmental processes to indirect encoding-based algorithms (Stanley and Miikkulainen, 2003; Mattiussi and Floreano, 2007; Suchorzewski, 2011; Kitano, 1990; Gruau, 1992). HyperNEAT by Stanley et al. (2009) and the variants that followed the original algorithm (Risi and Stanley, 2012a,b; D'Ambrosio et al., 2014) are powerful approaches to controller synthesis that employ an evolved generative encoding called Compositional Pattern Producing Network (CPPN) (Stanley, 2007). CPPNs are versatile, and have been used to synthesise different types of controllers (Tarapore and Mouret, 2015), such as open-loop and closed-loop central pattern generators (Ijspeert,

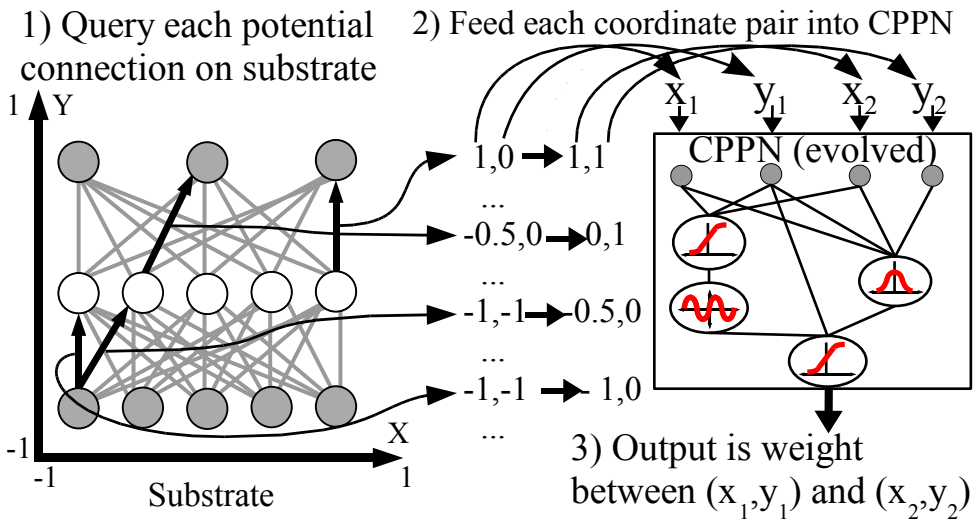


Figure A.1: HyperNEAT connectivity patterns production. Neurons in the ANN are assigned coordinates that range from -1 to 1 in all dimensions of a *substrate*. The weight of each potential connection in the substrate is determined by querying the CPPN. The dark directed lines in the substrate represent a sample of connections that are queried. For each query, the CPPN takes as input the positions of two neurons, and outputs the weight of the connection between them. As a result, CPPNs can produce regular patterns of connections. From Risi and Stanley (2012a). © MIT Press.

2008), single-unit pattern generators (Morse et al., 2013), and artificial neural networks. This discussion focuses on the optimisation of artificial neural networks because they are the prevalent control paradigm in evolutionary robotics (Nelson et al., 2009).

Formally, CPPNs are a composition of functions that encode the weight patterns of an ANN, as shown in Figure A.1. In conceptual terms, CPPNs are a variant of ANNs. The main difference between the two types of networks is that CPPNs rely on multiple different activation functions, which are composed to produce a pattern when CPPNs are queried over some input geometry (e.g. a two-dimensional coordinate space). Each activation function in CPPNs represents a specific regularity such as symmetry, repetition, or repetition with variation (Stanley, 2007). For instance, a periodic function such as sine creates repetition, while a Gaussian function enables left-right symmetry.

In HyperNEAT, CPPNs produce connectivity patterns by interpreting spatial patterns generated within a hypercube as connectivity patterns in a lower-dimensional space. Neurons exist at *locations*, and one CPPN uses the coordinates of pairs of neurons to compute the connection weights for the entire network, as illustrated in Figure A.1. Artificial neurons are thus *spatially sensitive*. Because the connection weights between neurons are a function of the geometric position of such neurons, HyperNEAT can exploit the neural *topography* and not just the topology, and is able to automatically find the geometric aspects of a task (Clune et al., 2011; Stanley et al., 2009). In addition, the fact that CPPNs are *evolved* encodings approximates nature where the mapping from genotype to phenotype itself is also subject to evolution.

In evolutionary robotics domains, HyperNEAT yields important advantages over other algorithms. Given its ability to discover the task geometry, HyperNEAT has been shown capable of: (i) correlating the internal geometry of the ANN with the placement of robot sensors and actuators, and of generalising the learnt geometric principles to create functional larger-scale ANNs with additional inputs and outputs (Stanley et al., 2009;

D'Ambrosio and Stanley, 2007), (ii) representing homogeneous and heterogeneous controllers for large groups of robots as a function of the control policy geometry, that is, the relationship between the role of the robots and their position in the group (D'Ambrosio et al., 2010; D'Ambrosio and Stanley, 2013), and (iii) evolving high-performing controllers for simulated quadruped robots by exploiting regularities such as four-way symmetry, wherein all legs of the robots continuously move in unison with both front-back symmetry and left-right symmetry (Clune et al., 2011). HyperNEAT's performance in the evolution of gaits for four-legged robots is particularly noteworthy because using other approaches, researchers typically need to manually identify the underlying task regularities and then force the encoding to exploit them (Clune et al., 2011; Stanley et al., 2009; Gauci and Stanley, 2008).

Even though HyperNEAT is able to exploit task geometry, its performance decreases on tasks that contain irregularities (Clune et al., 2011; Clune, 2010). For example, in the evolution of gaits for quadruped robots, if robots have faulty joints then HyperNEAT's ability to evolve coordinated behaviours is limited. As the number of faulty joints increases, HyperNEAT's performance continuously decreases and becomes statistically indistinguishable from that of its direct encoding counterpart NEAT (Stanley and Miikkulainen, 2002), which is both blind to task geometry and susceptible to the task dimensionality (800 parameters to be optimised) (Clune et al., 2011).

A.2 Hybridising Encodings

Clune et al. (2011) recently proposed a hybridisation of indirect and direct encodings, an algorithm called *switch-HybrID* to address HyperNEAT's ineffectiveness when irregularity is required. Switch-HybrID first evolves with HyperNEAT, and then switches to NEAT (Stanley and Miikkulainen, 2002) after a fixed, predefined number of generations. When the switch is made, each HyperNEAT ANN is translated to a NEAT genome. The evolutionary process then continues with NEAT until the end of the experiment. In Clune et al. (2011), switch-HybrID was shown to outperform HyperNEAT in three tasks: (i,ii) two diagnostic tasks called *the target weights problem* and *the bit mirroring problem*, in which the degree of regularity can be varied, and (iii) in gait learning for quadruped robots, with and without faulty joints. The key idea is that because switch-HybrID is able to make subtle adjustments to otherwise regular patterns, it can account for irregularities, including faulty joints in quadruped robots (Clune et al., 2011).

The success of switch-HybrID suggests that indirect encodings may be more effective not as stand-alone algorithms, but in combination with a refining process that adjusts regular patterns in irregular ways. However, a key disadvantage of switch-HybrID is that the switch from HyperNEAT to NEAT is only made after a number of generations defined by the experimenter elapse, meaning that evolution is limited to first exploring the solution space according to the properties of HyperNEAT and then to the properties of NEAT. Similarly to devising optimal stopping criteria for evolutionary algorithms based on, for example, search space exploration, objective convergence, or population convergence (Aytug and Koehler, 2000; Jain et al., 2001; Goel and Stander, 2010), defining an appropriate switch point is a non-trivial task that requires domain-specific knowledge. In addition, the switch-point criterion limits the applicability of switch-HybrID in more open-ended domains such as when controllers are evolved online and must adapt to changing environmental conditions.

As an alternative to switch-HybrID, we have recently introduced the R-HybrID algorithm in which ANN-

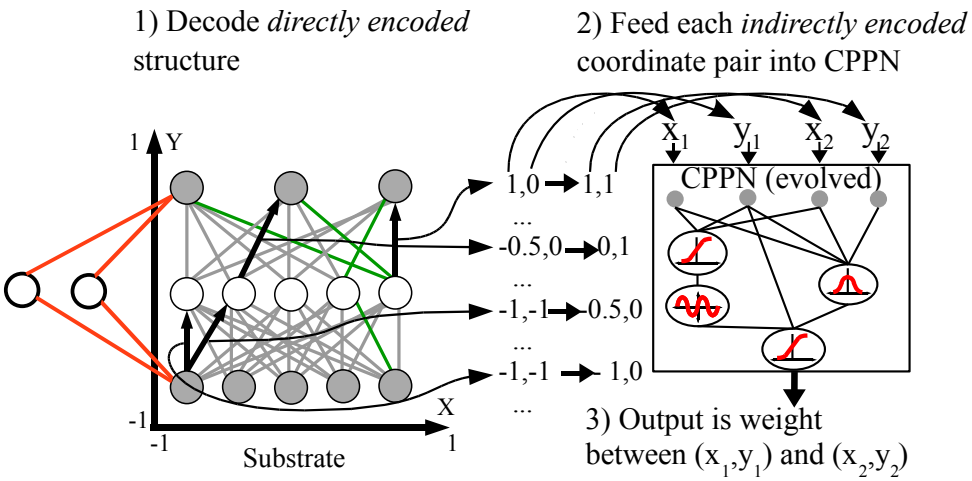


Figure A.2: R-HybrID connectivity patterns production. Samples of directly encoded and of indirectly encoded connections are represented in the substrate by the green undirected connections and by the black directed connections, respectively. Firstly, directly encoded connection weights are decoded. Secondly, indirectly encoded connection weights are decoded by querying the CPPN. Besides evolving the CPPN, R-HybrID can also add new directly encoded neurons and connections, represented in the left part of the substrate by the red undirected connections and corresponding neurons.

based controllers are partially indirectly encoded and partially directly encoded (Silva et al., 2015d). Specifically, genomes are composed of a direct encoding part, similar to NEAT, and an indirect encoding part, that is, a CPPN. Which parts of a given ANN are indirectly encoded via a CPPN or directly encoded is under evolutionary control. In this way, the evolutionary process has the potential to search for solutions across multiple ratios of indirect vs. direct encoding simultaneously, and to automatically find an appropriate encoding combination to solve the current task.

The decoding of R-HybrID genomes is shown in Figure A.2. Firstly, the directly encoded connection weights are decoded, if any. Secondly, the indirectly encoded connection weights are decoded by querying a CPPN. One other advantage of R-HybrID is that, because of the hybrid genomic representation, the evolutionary process can simultaneously optimise and complexify the CPPNs *and* the directly encoded structure. Because R-HybrID inherits NEAT and HyperNEAT's speciation dynamics, the algorithm can effectively maintain a variety of ANNs with differing complexities and encodings simultaneously over the course of evolution. R-HybrID was shown to: (i) outperform both NEAT and switch-HybrID, and to provide results comparable to HyperNEAT in the evolution of regular, large-scale controllers for a high-dimensional visual discrimination task (Stanley et al., 2009) that requires geometric principles to be evolved, and (ii) to typically outperform HyperNEAT, NEAT, and switch-HybrID in the coupled inverted pendulum (Hamann et al., 2011), a benchmark for modular robotics scenarios, and in a task inspired by the AX-CPT working memory test (Servan-Schreiber et al., 1996) that requires accumulating neural structure for cognitive behaviour to emerge (Lehman and Miikkulainen, 2014; Ollion et al., 2012).

Bibliography

Angeline, P., Saunders, G., and Pollack, J. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54 –65.

Asmuni, H., Burke, E., Garibaldi, J., and McCollum, B. (2004). Fuzzy multiple heuristic orderings for examination timetabling. In *5th International Conference on the Practice and Theory of Automated Timetabling*, pages 334–353. Springer, Berlin, Germany.

Aytug, H. and Koehler, G. J. (2000). New stopping criterion for genetic algorithms. *European Journal of Operational Research*, 126(3):662–674.

Baele, G., Bredeche, N., Haasdijk, E., Maere, S., Michiels, N., Van De Peer, Y., Schmickl, T., Schwarzer, C., and Thenius, R. (2009). Open-ended on-board evolutionary robotics for robot swarms. In *IEEE Congress on Evolutionary Computation*, pages 1123–1130. IEEE Press, Piscataway, NJ.

Bahgeçi, E. and Sahin, E. (2005). Evolving aggregation behaviors for swarm robotic systems: A systematic case study. In *IEEE Swarm Intelligence Symposium*, pages 333–340. IEEE Press, New York, NY.

Bailey, C., Giustetto, M., Huang, Y.-Y., Hawkins, R., and Kandel, E. (2000). Is heterosynaptic modulation essential for stabilizing Hebbian plasticity and memory? *Nature Reviews Neuroscience*, 1(1):11–20.

Barricelli, N. A. (1962). Numerical testing of evolution theories. *Acta Biotheoretica*, 16(1-2):69–98.

Bedini, S. A. (1964). The role of automata in the history of technology. *Technology and Culture*, 5(1):24–42.

Bekey, G. and Yuh, J. (2008). The status of robotics. *IEEE Robotics & Automation Magazine*, 15(1):80–86.

Bellingham, J. G. and Rajan, K. (2007). Robotics in remote and hostile environments. *Science*, 318(5853):1098–1102.

Bentley, P. and Kumar, S. (1999). Three ways to grow designs: A comparison of evolved embryogenies for a design problem. In *1st Genetic and Evolutionary Computation Conference*, pages 35–43. ACM Press, New York, NY.

Bianco, R. and Nolfi, S. (2004). Toward open-ended evolutionary robotics: evolving elementary robotic units able to self-assemble and self-reproduce. *Connection Science*, 16(4):227–248.

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *4th Genetic and Evolutionary Computation Conference*, pages 11–18. Morgan Kauffmann, San Francisco, CA.

Blakesley, R. E. (2008). *Parametric Control of Familywise Error Rates with Dependent ρ-values*. PhD thesis, University of Pittsburgh, Pennsylvania, PA.

Blynel, J. and Floreano, D. (2002). Levels of dynamics and adaptive behavior in evolutionary neural controllers. In *7th International Conference on Simulation of Adaptive Behavior*, pages 272–281. MIT Press, Cambridge, MA.

Boers, E. and Kuiper, H. (1992). Biological metaphors and the design of modular artificial neural networks. Master's thesis, Departments of Computer Science and Experimental and Theoretical Psychology at Leiden University, The Netherlands.

Bongard, J. (2002). Evolving modular genetic regulatory networks. In *IEEE Congress on Evolutionary Computation*, pages 1872–1877. IEEE Press, Piscataway, NJ.

Bongard, J. (2011). Morphological change in machines accelerates the evolution of robust behavior. *Proceedings of the National Academy of Sciences*, 108(4):1234–1239.

Bongard, J. (2013). Evolutionary robotics. *Communications of the ACM*, 56(8):74–83.

Bongard, J., Zykov, V., and Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121.

Braitenberg, V. (1984). *Vehicles: experiments in synthetic psychology*. MIT Press, Cambridge, MA.

Bredeche, N., Haasdijk, E., and Eiben, A. (2009). On-line, on-board evolution of robot controllers. In *9th International Conference on Artificial Evolution*, pages 110–121. Springer, New York, NY.

Bredeche, N., Montanier, J., Liu, W., and Winfield, A. (2012). Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. *Mathematical and Computer Modelling of Dynamical Systems*, 18(1):101–129.

Brooks, R. A. (1992). Artificial life and real robots. In *1st European Conference on Artificial Life*, pages 3–10. MIT Press, Cambridge, MA.

Brooks, R. A. and Matarić, M. J. (1993). *Real Robots, Real Learning Problems*, pages 193–213. Springer, Boston, MA.

Burke, E., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724.

Burke, E., McCollum, B., Meisels, A., Petrovic, S., and Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192.

Camazine, S., Deneubourg, J.-L., Franks, N., Sneyd, J., Theraulaz, G., and Bonabeau, E. (2001). *Self-Organization in Biological Systems*. Princeton University Press, Princeton, NJ.

Cao, Y., Fukunaga, A., and Kahng, A. (1997). Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):1–23.

Capdepuy, P., Polani, D., and Nehaniv, C. (2007). Maximization of potential information flow as a universal utility for collective behaviour. In *IEEE Symposium on Artificial Life*, pages 207–213. IEEE Press, Piscataway, NJ.

Carlson, J., Murphy, R., and Nelson, A. (2004). Follow-up analysis of mobile robot failures. In *IEEE International Conference on Robotics and Automation*, pages 4987–4994. IEEE Computer Society Press, Los Alamitos, CA.

Cazenille, L., Bredeche, N., Hamann, H., and Stradner, J. (2012). Impact of neuron models and network structure on evolving modular robot neural network controllers. In *14th Genetic and Evolutionary Computation Conference*, pages 89–96. ACM Press, New York, NY.

Chiel, H. J. and Beer, R. D. (1997). The brain has a body: adaptive behavior emerges from interactions of nervous system, body and environment. *Trends in Neurosciences*, 20(12):553–557.

Christensen, A. L., O'Grady, R., and Dorigo, M. (2009). From fireflies to fault-tolerant swarms of robots. *IEEE Transactions on Evolutionary Computation*, 13(4):754–766.

Christensen, A. L., Oliveira, S. M., Postolache, O., De Oliveira, M. J., Sargento, S., Santana, P., Nunes, L., Velez, F., Sebastiao, P., Costa, V., Duarte, M., Gomes, J., Rodrigues, T., and Silva, F. (2015). Design of communication and control for swarms of aquatic surface drones. In *7th International Conference on Agents and Artificial Intelligence*, pages 548–555. SCITEPRESS, Lisbon, Portugal.

Clark, A. (1998). *Being there: Putting brain, body, and world together again.* MIT Press, Cambridge, MA.

Clune, J. (2010). *Evolving artificial neural networks with generative encodings inspired by developmental biology.* PhD thesis, Michigan State University, East Lansing, MI.

Clune, J., Stanley, K. O., Pennock, R., and Ofria, C. (2011). On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 15(3):346–367.

Cowling, P., Kendall, G., and Soubeiga, E. (2001). A hyperheuristic approach to scheduling a sales summit. In *3rd International Conference on the Practice and Theory of Automated Timetabling*, pages 176–190. Springer, Berlin, Germany.

Cuccu, G. and Gomez, F. (2011). When novelty is not enough. In *13th European Conference on the Applications of Evolutionary Computation*, pages 234–243. Springer, Berlin, Germany.

Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B. (2015). Robots that can adapt like animals. *Nature*, 521(7553):503–507.

Cully, A. and Mouret, J.-B. (2013). Behavioral repertoire learning in robotics. In *15th Genetic and Evolutionary Computation Conference*, pages 175–182. ACM Press, New York, NY.

Cully, A. and Mouret, J.-B. (2015). Evolving a behavioral repertoire for a walking robot. *Evolutionary Computation*, 24(1):59–88.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314.

D'Ambrosio, D., Gauci, J., and Stanley, K. O. (2014). HyperNEAT: The first five years. In *Growing Adaptive Machines*, volume 557 of *Studies in Computational Intelligence*, chapter 5, pages 159–185. Springer, Berlin, Germany.

D'Ambrosio, D., Lehman, J., Risi, S., and Stanley, K. O. (2010). Evolving policy geometry for scalable multiagent learning. In *9th International Conference on Autonomous Agents and Multiagent Systems*, pages 731–738. IFAAMAS, Richland, SC.

D'Ambrosio, D. and Stanley, K. O. (2007). A novel generative encoding for exploiting neural network sensor and output geometry. In *9th Genetic and Evolutionary Computation Conference*, pages 974–981. ACM Press, New York, NY.

D'Ambrosio, D. and Stanley, K. O. (2013). Scalable multiagent learning through indirect encoding of policy geometry. *Evolutionary Intelligence*, 6(1):1–26.

Dawkins, R. (1976). *The Selfish Gene*. Oxford University Press, Oxford, UK.

de Jong, K. (2006). *Evolutionary Computation: A Unified Approach*. MIT Press, Cambridge, MA.

de Jong, K. and Potter, M. (1995). Evolving complex structures via cooperative coevolution. In *4th Annual Conference on Evolutionary Programming*, pages 307–317. MIT Press, Cambridge, MA.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Doncieux, S. and Mouret, J.-B. (2014). Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, 7(2):71–93.

Doncieux, S., Mouret, J.-B., Bredeche, N., and Padois, V. (2011). Evolutionary robotics: Exploring new horizons. In *New Horizons in Evolutionary Robotics*, volume 341 of *Studies in Computational Intelligence*, chapter 1, pages 3–25. Springer, Berlin, Germany.

Duarte, M., Silva, F., Rodrigues, T., Oliveira, S. M., and Christensen, A. L. (2014). JBotEvolver: A versatile simulation platform for evolutionary robotics. In *14th International Conference on the Synthesis and Simulation of Living Systems*, pages 210–211. MIT Press, Cambridge, MA.

Durbin, R. and Rumelhart, D. E. (1989). Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, 1(1):133–142.

Eiben, A., Haasdijk, E., and Bredeche, N. (2010a). Embodied, on-line, on-board evolution for autonomous robotics. In *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*, volume 7 of *Cognitive Systems Monographs*, chapter 7, pages 361–382. Springer, New York, NY.

Eiben, A., Karafotias, G., and Haasdijk, E. (2010b). Self-adaptive mutation in on-line, on-board evolutionary robotics. In *Workshops of the 4th IEEE International Conference on Self-Adaptive Self-Organizing Systems*, pages 147–152. IEEE Press, Piscataway, NJ.

Eiben, A., Michalewicz, Z., Schoenauer, M., and Smith, J. (2007). Parameter control in evolutionary algorithms. In *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, chapter 2, pages 19–46. Springer, Berlin, Germany.

Ellefsen, K. O., Mouret, J.-B., and Clune, J. (2015). Neural modularity helps organisms evolve to learn new skills without forgetting old skills. *PLoS Computational Biology*, 11(4):e1004128.

Elman, J. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

Farinelli, A., Iocchi, L., and Nardi, D. (2004). Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics B*, 34(5):2015–2028.

Fisher, R. A. (1925). *Statistical Methods For Research Workers*. Oliver & Boyd, Edinburgh, UK.

Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62.

Floreano, D. and Keller, L. (2010). Evolution of adaptive behaviour by means of Darwinian selection. *PLoS Biology*, 8(1):e1000292.

Floreano, D. and Mondada, F. (1994). Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In *3rd International Conference on Simulation of Adaptive Behavior*, pages 421–430. MIT Press, Cambridge, MA.

Floreano, D. and Mondada, F. (1996). Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics B*, 26(3):396–407.

Floreano, D. and Urzelai, J. (2000). Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, 13(4–5):431–443.

Floreano, D., Zufferey, J.-C., and Nicoud, J.-D. (2005). From wheels to wings with evolutionary spiking circuits. *Artificial Life*, 11(1-2):121–138.

Fogel, D. (2006). Nils Barricelli – artificial life, coevolution, self-adaptation. *IEEE Computational Intelligence Magazine*, 1(1):41–45.

Fraser, A. (1957). Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Sciences*, 10:484–491.

Funahashi, K. and Nakamura, Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806.

García-Sánchez, P., Eiben, A., Haasdijk, E., Weel, B., and Merelo-Guervós, J.-J. (2012). Testing diversity-enhancing migration policies for hybrid on-line evolution of robot controllers. In *14th European Conference on the Applications of Evolutionary Computation*, pages 52–62. Springer, Berlin, Germany.

Gates, B. (2007). A robot in every home. *Scientific American*, 296(1):58–65.

Gauci, J. and Stanley, K. O. (2008). A case study on the critical role of geometric regularity in machine learning. In *23rd AAAI Conference on Artificial Intelligence*, pages 628–633. AAAI Press, Menlo Park, CA.

Gerstner, W. and Kistler, W. M. (2002). *Spiking neuron models: Single neurons, populations, plasticity.* Cambridge University Press, Cambridge, UK.

Goel, T. and Stander, N. (2010). A non-dominance-based online stopping criterion for multi-objective evolutionary algorithms. *International Journal for Numerical Methods in Engineering*, 84(6):661–684.

Goldberg, D., Deb, K., and Korb, B. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493–530.

Goldsby, H. and Cheng, B. (2010). Automatically discovering properties that specify the latent behavior of UML models. In *13th International Conference on Model Driven Engineering Languages and Systems*, pages 316–330. Springer, Berlin, Germany.

Gomes, J. and Christensen, A. L. (2013). Generic behaviour similarity measures for evolutionary swarm robotics. In *15th Genetic and Evolutionary Computation Conference*, pages 199–206. ACM Press, New York, NY.

Gomes, J., Urbano, P., and Christensen, A. L. (2012). Progressive minimal criteria novelty search. In *13th Ibero-American Conference on Artificial Intelligence*, pages 281–290. Springer, Berlin, Germany.

Gomez, F. and Miikkulainen, R. (1997). Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3–4):317–342.

Gomez, F. and Miikkulainen, R. (2003). *Robust non-linear control through neuroevolution.* PhD thesis, University of Texas, Austin, TX.

Gould, S. (2002). *The Structure of Evolutionary Theory.* Belknap Press, Cambridge, MA.

Grobler, J., Engelbrecht, A. P., Kendall, G., and Yadavalli, V. (2012). Investigating the use of local search for improving meta-hyper-heuristic performance. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE Press, Piscataway, NJ.

Groß, R. and Dorigo, M. (2009). Towards group transport by swarms of robots. *International Journal of Bio-Inspired Computation*, 1(1–2):1–13.

Gruau, F. (1992). Genetic synthesis of boolean neural networks with a cell rewriting developmental process. In *International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 55–74. IEEE Press, Piscataway, NJ.

Gutiérrez, A., Campo, A., Dorigo, M., Amor, D., Magdalena, L., and Monasterio-Huelin, F. (2008). An open localization and local communication embodied sensor. *Sensors*, 8(11):7545–7563.

Haasdijk, E., Atta-ul Qayyum, A., and Eiben, A. (2011). Racing to improve on-line, on-board evolutionary robotics. In *13th Genetic and Evolutionary Computation Conference*, pages 187–194, ACM Press, New York, NY.

Haasdijk, E. and Bredeche, N. (2013). Controlling task distribution in MONEE. In *12th European Conference on Artificial Life*, pages 671–678. MIT Press, Cambridge, MA.

Haasdijk, E., Eiben, A., and Karafotias, G. (2010). On-line evolution of robot controllers by an encapsulated evolution strategy. In *IEEE Congress on Evolutionary Computation*, pages 1–7. IEEE Press, Piscataway, NJ.

Haasdijk, E., Smit, S. K., and Eiben, A. E. (2012). Exploratory analysis of an on-line evolutionary algorithm in simulated robots. *Evolutionary Intelligence*, 5(4):213–230.

Hamann, H., Schmickl, T., and Crailsheim, K. (2011). Coupled inverted pendulums: a benchmark for evolving decentral controllers in modular robotics. In *13th Genetic and Evolutionary Computation Conference*, pages 195–202. ACM Press, New York, NY.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195.

Harp, S., Samad, T., and Guha, A. (1989). Towards the genetic synthesis of neural network. In *3rd International Conference on Genetic Algorithms*, pages 360–369. Morgan Kaufmann, San Francisco, CA.

Harvey, I. (2011). The microbial genetic algorithm. In *10th European Conference on Artificial Life*, pages 126–133. Springer, Berlin, Germany.

Harvey, I., Di Paolo, E., Wood, R., Quinn, M., and Tuci, E. (2005). Evolutionary robotics: A new scientific tool for studying cognition. *Artificial Life*, 11(1–2):79–98.

Harvey, I., Husbands, P., and Cliff, D. (1993). Issues in evolutionary robotics. In *2nd International Conference on Simulation of Adaptive Behavior*, pages 364–373. MIT Press, Cambridge, MA.

Harvey, I., Husbands, P., and Cliff, D. (1994). Seeing the light: Artificial evolution, real vision. In *3rd International Conference on Simulation of Adaptive Behavior*, pages 392–401. MIT Press, Cambridge, MA.

Harvey, I., Husbands, P., Cliff, D., Thompson, A., and Jakobi, N. (1997). Evolutionary Robotics: the Sussex approach. *Robotics and Autonomous Systems*, 20(2–4):205–224.

Haykin, S. (1999). *Neural Networks: a Comprehensive Foundation*. Prentice-Hall, Englewood Cliffs, NJ.

Heinerman, J., Zonta, A., Haasdijk, E., and Eiben, A. (2016). On-line evolution of foraging behaviour in a population of real robots. In *19th European Conference on the Applications of Evolutionary Computation*, pages 198–212. Springer International Publishing, Switzerland.

Herculano-Houzel, S. (2009). The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in Human Neuroscience*, 3(31).

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

Holland, J. (1962). Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9(3):297–314.

Hommel, G. (1988). A stagewise rejective multiple test procedure based on a modified Bonferroni test. *Biometrika*, 75(2):383–386.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

Huijsman, R., Haasdijk, E., and Eiben, A. (2011). An on-line on-board distributed algorithm for evolutionary robotics. In *10th International Conference on Artificial Evolution*, pages 73–84. Springer, Berlin, Germany.

Husbands, P., Harvey, I., Cliff, D., and Miller, G. (1997). Artificial evolution: a new path for artificial intelligence? *Brain and Cognition*, 34(1):130–159.

Igel, C. (2003). Neuroevolution for reinforcement learning using evolution strategies. In *IEEE Congress on Evolutionary Computation*, pages 2588–2595. IEEE Press, Piscataway, NJ.

Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 21(4):642–653.

Jain, B. J., Pohlheim, H., and Wegener, J. (2001). On termination criteria of evolutionary algorithms. In *3rd Genetic and Evolutionary Computation Conference*, page 768. ACM Press, New York, NY.

Jakobi, N. (1997). Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive Behavior*, 6(2):325–368.

Jones, C. and Mataríc, M. (2006). Behavior-based coordination in multi-robot systems. In *Autonomous Mobile Robots: Sensing, Control, Decision Making and Applications*, volume 22 of *Automation and Control Engineering*, chapter 14, pages 549–569. CRC Press, Boca Raton, FL.

Jordan, M. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. In *8th Annual Meeting of the Cognitive Science Society*, pages 531–546. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ.

Kantschik, W., Dittrich, P., Brameier, M., and Banzhaf, W. (1999). Empirical analysis of different levels of meta-evolution. In *IEEE Congress on Evolutionary Computation*, pages 2086–2093. IEEE Press, Piscataway, NJ.

Karafotias, G., Haasdijk, E., and Eiben, A. (2011). An algorithm for distributed on-line, on-board evolutionary robotics. In *13th Genetic and Evolutionary Computation Conference*, pages 171–178. ACM Press, New York, NY.

Kassahun, Y. and Sommer, G. (2005). Efficient reinforcement learning through evolutionary acquisition of neural topologies. In *13th European Symposium on Artificial Neural Networks*, pages 259–266. ISBN 2-930307-05-6.

Katz, P. (1999). *Beyond Neurotransmission: Neuromodulation and its importance for information processing.* Oxford University Press, Oxford, UK.

Kim, Y.-H. and Moon, B.-R. (2003). New usage of Sammon's mapping for genetic visualization. In *5th Genetic and Evolutionary Computation Conference*, pages 1136–1147. Springer, Berlin, Germany.

Kistemaker, S. and Whiteson, S. (2011). Critical factors in the performance of novelty search. In *13th Genetic and Evolutionary Computation Conference*, pages 965–972. ACM Press, New York, NY.

Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4(4):461–476.

Kittler, J. and Young, P. (1973). A new approach to feature selection based on the Karhunen-Loeve expansion. *Pattern Recognition*, 5(4):335–352.

Klyubin, A. S., Polani, D., and Nehaniv, C. L. (2005a). All else being equal be empowered. In *8th European Conference on Artificial Life*, pages 744–753. Springer, Berlin, Germany.

Klyubin, A. S., Polani, D., and Nehaniv, C. L. (2005b). Empowerment: A universal agent-centric measure of control. In *IEEE Congress on Evolutionary Computation*, pages 128–135. IEEE Press, Piscataway, NJ.

Knowles, J. D., Watson, R. A., and Corne, D. W. (2001). Reducing local optima in single-objective problems by multi-objectivization. In *1st International Conference on Evolutionary Multi-criterion Optimization*, pages 269–283. Springer, Berlin, Germany.

Koch, C. (2004). *Biophysics of computation: information processing in single neurons*. Oxford University Press, Oxford, NY.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69.

König, L., Mostaghim, S., and Schmeck, H. (2009). Decentralized evolution of robotic behavior using finite state machines. *International Journal of Intelligent Computing and Cybernetics*, 2(4):695–723.

Koos, S., Mouret, J.-B., and Doncieux, S. (2013). The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145.

Kube, C. and Zhang, H. (1993). Collective robotics: From social insects to robots. *Adaptive Behavior*, 2(2):189–218.

Laredo, J., Eiben, A., van Steen, M., and Merelo, J. (2010). EvAg: a scalable peer-to-peer evolutionary algorithm. *Genetic Programming and Evolvable Machines*, 11(2):227–246.

Lehman, J. (2012). *Evolution through the Search for Novelty*. PhD thesis, University of Central Florida, Orlando, FL.

Lehman, J. and Miikkulainen, R. (2014). Overcoming deception in evolution of cognitive behaviors. In *16th Genetic and Evolutionary Computation Conference*, pages 185–192. ACM Press, New York, NY.

Lehman, J., Risi, S., D'Ambrosio, D., and Stanley, K. O. (2013a). Encouraging reactivity to create robust machines. *Adaptive Behavior*, 21(6):484–500.

Lehman, J. and Stanley, K. O. (2008). Exploiting open-endedness to solve problems through the search for novelty. In *11th International Conference on Simulation and Synthesis of Living Systems*, pages 329–336. MIT Press, Cambridge, MA.

Lehman, J. and Stanley, K. O. (2010). Revising the evolutionary computation abstraction: minimal criteria novelty search. In *12th Genetic and Evolutionary Computation Conference*, pages 103–110. ACM Press, New York, NY.

Lehman, J. and Stanley, K. O. (2011a). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223.

Lehman, J. and Stanley, K. O. (2011b). Evolving a diversity of virtual creatures through novelty search and local competition. In *13th Genetic and Evolutionary Computation Conference*, pages 211–218. ACM Press, New York, NY.

Lehman, J. and Stanley, K. O. (2011c). Novelty search and the problem with objectives. In *Genetic Programming Theory and Practice IX*, Genetic and Evolutionary Computation, chapter 3, pages 37–56. Springer, Berlin, Germany.

Lehman, J., Stanley, K. O., and Miikkulainen, R. (2013b). Effective diversity maintenance in deceptive domains. In *15th Genetic and Evolutionary Computation Conference*, pages 215–222. ACM Press, New York, NY.

Lewis, M. A., Fagg, A. H., and Solidum, A. (1992). Genetic programming approach to the construction of a neural network for control of a walking robot. In *IEEE International Conference on Robotics and Automation*, pages 2618–2623. IEEE Press, Piscataway, NJ.

Liapis, A., Yannakakis, G. N., and Togelius, J. (2013). Enhancements to constrained novelty search: Two-population novelty search for generating game content. In *15th Genetic and Evolutionary Computation Conference*, pages 343–350. ACM Press, New York, NY.

Lin, T., Horne, B., and Giles, C. (1996). How embedded memory in recurrent neural network architectures helps learning long-term temporal dependencies. Technical report, University of Maryland, College Park, MD.

Lipson, H. and Pollack, J. (2000). Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–978.

Lobo, F. G. (2000). *The Parameter-Less Genetic Algorithm: Rational and Automated Parameter Selection for Simplified Genetic Algorithm Operation*. PhD thesis, Universidade Nova de Lisboa, Lisbon, Portugal.

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671.

Maron, O. and Moore, A. W. (1997). The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11(1):193–225.

Matarić, M. and Cliff, D. (1996). Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems*, 19(1):67–83.

Mattiussi, C. and Floreano, D. (2007). Analog genetic encoding for the evolution of circuits and networks. *IEEE Transactions on Evolutionary Computation*, 11(5):596–607.

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133.

McFarland, D. and Bösser, T. (1993). *Intelligent behavior in animals and robots*. MIT Press, Cambridge, MA.

Meyer, J.-A., Husbands, P., and Harvey, I. (1998). Evolutionary robotics: A survey of applications and problems. In *1st European Workshop on Evolutionary Robotics*, pages 1–21. Springer, Berlin, Germany.

Miglino, O., Lund, H., and Nolfi, S. (1995). Evolving mobile robots in simulated and real environments. *Artificial Life*, 2(4):417–434.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J., Floreano, D., and Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In *9th Conference on Autonomous Robot Systems and Competitions*, pages 59–65. IPCB, Castelo Branco, Portugal.

Mondada, F. and Floreano, D. (1995). Evolution of neural control structures: some experiments on mobile robots. *Robotics and Autonomous Systems*, 16(2):183–195.

Mondada, F., Pettinaro, G., Guignard, A., Kwee, I., Floreano, D., Deneubourg, J., Nolfi, S., Gambardella, L., and Dorigo, M. (2004). Swarm-bot: A new distributed robotic concept. *Autonomous Robots*, 17(2–3):193–221.

Montanier, J.-M. (2013). *Environment-driven Distributed Evolutionary Adaptation for Collective Robotic Systems*. PhD thesis, Université Paris Sud-Paris XI, France.

Montanier, J.-M. and Bredeche, N. (2011). Embedded evolutionary robotics: The (1+1)-restart-online adaptation algorithm. In *New Horizons in Evolutionary Robotics*, volume 341 of *Studies in Computational Intelligence*, chapter 11, pages 155–169. Springer, Berlin, Germany.

Moriarty, D. and Miikkulainen, R. (1996). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22(1):11–32.

Morse, G., Risi, S., Snyder, C. R., and Stanley, K. O. (2013). Single-unit pattern generators for quadruped locomotion. In *15th Genetic and Evolutionary Computation Conference*, pages 719–726. ACM Press, New York, NY.

Mouret, J.-B. (2011). Novelty-based multiobjectivization. In *New Horizons in Evolutionary Robotics*, volume 341 of *Studies in Computational Intelligence*, chapter 10, pages 139–154. Springer, Berlin, Germany.

Mouret, J.-B. and Clune, J. (2015). Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*.

Mouret, J.-B. and Doncieux, S. (2009). Using behavioral exploration objectives to solve deceptive problems in neuro-evolution. In *11th Genetic and Evolutionary Computation Conference*, pages 627–634. ACM Press, New York, NY.

Mouret, J.-B. and Doncieux, S. (2012). Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary Computation*, 20(1):91–133.

Murre, J. M. J. and Sturdy, D. P. F. (1995). The connectivity of the brain: multi-level quantitative analysis. *Biological Cybernetics*, 73(6):529–545.

Muscettola, N., Nayak, P. P., Pell, B., and Williams, B. C. (1998). Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1):5–47.

Naredo, E. and Trujillo, L. (2013). Searching for novel clustering programs. In *15th Genetic and Evolutionary Computation Conference*, pages 1093–1100. ACM Press, New York, NY.

Nelson, A., Barlow, G., and Doitsidis, L. (2009). Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370.

Nolfi, S. (1998). Evolutionary robotics: Exploiting the full power of self-organization. *Connection Science*, 10(3–4):167–184.

Nolfi, S. and Floreano, D. (2000). *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT Press, Cambridge, MA.

Nolfi, S. and Floreano, D. (2002). Synthesis of autonomous robots through evolution. *Trends in Cognitive Sciences*, 6(1):31–37.

Nolfi, S., Floreano, D., Miglino, O., and Mondada, F. (1994). How to evolve autonomous robots: Different approaches in evolutionary robotics. In *4th International Workshop on Synthesis and Simulation of Living Systems*, pages 190–197. MIT Press, Cambridge, MA.

Noskov, N., Haasdijk, E., Weel, B., and Eiben, A. (2013). MONEE: Using parental investment to combine open-ended and task-driven evolution. In *15th European Conference on the Applications of Evolutionary Computation*, pages 569–578. Springer, Berlin, Germany.

Ollion, C., Pinville, T., and Doncieux, S. (2012). With a little help from selection pressures: evolution of memory in robot controllers. In *13th International Conference on Simulation and Synthesis of Living Systems*, pages 407–414. MIT Press, Cambridge, MA.

Omidvar, M. N. and Li, X. (2011). A comparative study of CMA-ES on large scale global optimisation. In *23rd Australasian Joint Conference on Artificial Intelligence*, pages 303–312. Springer, Berlin, Germany.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286.

Pillay, N. and Banzhaf, W. (2007). A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem. In *13th Portuguese Conference on Artificial Intelligence*, pages 223–234. Springer, Berlin, Germany.

Poli, R., Vanneschi, L., Langdon, W. B., and McPhee, N. F. (2010). Theoretical results in genetic programming: the next ten years? *Genetic Programming and Evolvable Machines*, 11(3-4):285–320.

Porr, B., Wörgötter, F., and Glasgow, G. (2006). Strongly improved stability and faster convergence of temporal sequence learning by utilising input correlations only. *Neural Computation*, 18(6):1380–1412.

Prieto, A., Becerra, J., Bellas, F., and Duro, R. (2010). Open-ended evolution as a means to self-organize heterogeneous multi-robot systems in real time. *Robotics and Autonomous Systems*, 58(12):1282–1291.

Prokopenko, M., Gerasimov, V., and Tanev, I. (2006). Evolving spatiotemporal coordination in a modular robotic system. In *9th International Conference on Simulation of Adaptive Behavior*, pages 558–569. Springer, Berlin, Germany.

Pugh, J. K., Soros, L., Szerlip, P. A., and Stanley, K. O. (2015). Confronting the challenge of quality diversity. In *17th Genetic and Evolutionary Computation Conference*, pages 967–974. ACM Press, New York, NY.

Pugh, J. K., Soros, L. B., and Stanley, K. O. (2016). Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40.

Quinn, M., Smith, L., Mayley, G., and Husbands, P. (2003). Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 361(1811):2321–2343.

Reisinger, J., Stanley, K. O., and Miikkulainen, R. (2004). Evolving reusable neural modules. In *6th Genetic and Evolutionary Computation Conference*, pages 69–81. Springer, Berlin, Germany.

Ring, M. (1994). *Continual Learning in Reinforcement Environments*. PhD thesis, University of Texas, Austin, TX.

Risi, S. (2012). *Towards Evolving More Brain-Like Artificial Neural Networks*. PhD thesis, University of Central Florida, Orlando, FL.

Risi, S. and Stanley, K. O. (2012a). An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial Life*, 18(4):331–363.

Risi, S. and Stanley, K. O. (2012b). A unified approach to evolving plasticity and neural geometry. In *International Joint Conference on Neural Networks*, pages 1–8. IEEE Press, Piscataway, NJ.

Rosheim, M. (2006). *Leonardo's Lost Robots*. Springer, Berlin, Germany.

Ross, P. and Marin-Blazquez, J. (2005). Constructive hyper-heuristics in class timetabling. In *IEEE Congress on Evolutionary Computation*, pages 1493–1500. IEEE Press, Piscataway, NJ.

Rossi, C., Russo, F., and Russo, F. (2009). *Automata (Towards Automation and Robots)*, volume 8 of *History of Mechanism and Machine Science*, chapter 16, pages 269–301. Springer, Netherlands.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ.

Ryser-Welch, P. and Miller, J. (2014). A review of hyper-heuristic frameworks. In *50th Annual Convention of the Society for the Study of Artificial Intelligence and the Simulation of Behaviour*. Available at `http://doc.gold.ac.uk/aisb50/`.

Saggie, K., Keinan, A., and Ruppin, E. (2004). Spikes that count: rethinking spikiness in neurally embedded systems. *Neurocomputing*, 58–60:303–311.

Sammon Jr., J. (1969). A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409.

Schaul, T., Sun, Y., Wierstra, D., Gomez, F., and Schmidhuber, J. (2011). Curiosity-driven optimization. In *IEEE Congress of Evolutionary Computation*, pages 1343–1349. IEEE Press, Piscataway, NJ.

Schmidhuber, J. (1997). Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873.

Schmitt, M. (2002). On the complexity of computing and learning with multiplicative neural networks. *Neural Computation*, 14(2):241–301.

Schwarzer, C., Schlachter, F., and Michiels, N. (2011). Online evolution in dynamic environments using neural networks in autonomous robots. *International Journal on Advances in Intelligent Systems*, 4(3–4):288–298.

Servan-Schreiber, D., Cohen, J. D., and Steingard, S. (1996). Schizophrenic deficits in the processing of context: A test of a theoretical model. *Archives of General Psychiatry*, 53(12):1105–1113.

Seys, C. W. and Beer, R. D. (2007). Genotype reuse more important than genotype size in evolvability of embodied neural networks. In *9th European Conference on Artificial Life*, pages 915–924. Springer, Berlin, Germany.

Siebel, N., Krause, J., and Sommer, G. (2007). Efficient learning of neural networks with evolutionary algorithms. *Pattern Recognition*, pages 466–475.

Silva, F., Christensen, A. L., and Correia, L. (2015a). Engineering online evolution of robot behaviour. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 2017–2018. IFAAMAS, Richland, SC.

Silva, F., Correia, L., and Christensen, A. L. (2013). Dynamics of neuronal models in online neuroevolution of robotic controllers. In *16th Portuguese Conference on Artificial Intelligence*, pages 90–101. Springer, Berlin, Germany.

Silva, F., Correia, L., and Christensen, A. L. (2014a). Speeding up online evolution of robotic controllers with macro-neurons. In *17th European Conference on the Applications of Evolutionary Computation*, pages 765–776. Springer, Berlin, Germany.

Silva, F., Correia, L., and Christensen, A. L. (2015b). A case study on the scalability of online evolution of robotic controllers. In *17th Portuguese Conference on Artificial Intelligence*, pages 189–200. Springer, Berlin, Germany.

Silva, F., Correia, L., and Christensen, A. L. (2015c). Modelling synchronisation in multirobot systems with cellular automata: Analysis of update methods and topology perturbations. In *Robots and Lattice Automata*, volume 13 of *Emergence, Complexity and Computation*, pages 267–293. Springer International Publishing, Switzerland.

Silva, F., Correia, L., and Christensen, A. L. (2015d). R-HybrID: Evolution of agent controllers with a hybridisation of indirect and direct encodings. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 735–744. IFAAMAS, Richland, SC.

Silva, F., Correia, L., and Christensen, A. L. (2016a). Evolutionary robotics. *Scholarpedia*, 11(7):33333.

Silva, F., Correia, L., and Christensen, A. L. (2016b). Leveraging online racing and population cloning in evolutionary multirobot systems. In *19th European Conference on the Applications of Evolutionary Computation*, pages 165–180. Springer International Publishing, Switzerland.

Silva, F., Correia, L., and Christensen, A. L. (2016c). Online hyper-evolution of controllers in multirobot systems. In *10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 11–20. IEEE Computer Society, Los Alamitos, CA.

Silva, F., Correia, L., and Christensen, A. L. (2017a). Evolutionary online behaviour learning and adaptation in real robots. *Royal Society Open Science*, 4:160938.

Silva, F., Correia, L., and Christensen, A. L. (2017b). Evolutionary online learning in multirobot systems. *AI Matters*, 3(1):23–24.

Silva, F., Correia, L., and Christensen, A. L. (2017c). Hyper-learning algorithms for online evolution of robot controllers. *ACM Transactions on Autonomous and Adaptive Systems*. To appear.

Silva, F., Duarte, M., Correia, L., Oliveira, S. M., and Christensen, A. L. (2016d). Open issues in evolutionary robotics. *Evolutionary Computation*, 24(2):205–236.

Silva, F., Duarte, M., Oliveira, S. M., Correia, L., and Christensen, A. L. (2014b). The case for engineering the evolution of robot controllers. In *14th International Conference on the Synthesis and Simulation of Living Systems*, pages 703–710. MIT Press, Cambridge, MA.

Silva, F., Urbano, P., Correia, L., and Christensen, A. L. (2015e). odNEAT: An algorithm for decentralised online evolution of robotic controllers. *Evolutionary Computation*, 23(3):421–449.

Silva, F., Urbano, P., Oliveira, S., and Christensen, A. L. (2012). odNEAT: An algorithm for distributed online, onboard evolution of robot behaviours. In *13th International Conference on the Simulation and Synthesis of Living Systems*, pages 251–258. MIT Press, Cambridge, MA.

Simões, E. and Barone, D. (2002). Predation: An approach to improving the evolution of real robots with a distributed evolutionary controller. In *IEEE International Conference on Robotics and Automation*, pages 664–669. IEEE Press, Piscataway, NJ.

Southan, C. (2004). Has the yo-yo stopped? An assessment of human protein-coding gene number. *Proteomics*, 4(6):1712–1726.

SPARC (2015). Robotics 2020 multi-annual roadmap – ICT-24. Technical report, euRobotics.

Sperati, V., Trianni, V., and Nolfi, S. (2008). Evolving coordinated group behaviours through maximisation of mean mutual information. *Swarm Intelligence*, 2(2-4):73–95.

Squyres, S., Arvidson, R., Bell, J. F., Brückner, J., Cabrol, N., Calvin, W., Carr, M., Christensen, P., Clark, B., Crumpler, L., et al. (2004). The Opportunity Rover's Athena science investigation at Meridiani Planum, Mars. *Science*, 306(5702):1698–1703.

Stanley, K. O. (2004). *Efficient Evolution of Neural Networks through Complexification.* PhD thesis, University of Texas, Austin, TX.

Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162.

Stanley, K. O., Bryant, B. D., and Miikkulainen, R. (2005). Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668.

Stanley, K. O., D'Ambrosio, D., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

Stanley, K. O. and Miikkulainen, R. (2003). A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130.

Stanley, K. O. and Miikkulainen, R. (2004). Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100.

Stix, G. (2006). Owning the stuff of life. *Scientific American*, 294(2):76–83.

Suchorzewski, M. (2011). Evolving scalable and modular adaptive networks with developmental symbolic encoding. *Evolutionary Intelligence*, 4(3):145–163.

Tanese, R. (1989). *Distributed Genetic Algorithms for Function Optimization.* PhD thesis, University of Michigan, Ann Arbor, MI.

Tarapore, D. and Mouret, J.-B. (2015). Evolvability signatures of generative encodings: Beyond standard performance benchmarks. *Information Sciences*, 313:43–61.

Trianni, V. (2006). *On the Evolution of Self-Organising Behaviours in a Swarm of Autonomous Robots.* PhD thesis, Université Libre de Bruxelles, Brussels, Belgium.

Trianni, V. (2008). *Evolutionary Swarm Robotics: Evolving Self-Organising Behaviours in Groups of Autonomous Robots*, volume 108 of *Studies in Computational Intelligence.* Springer, Berlin, Germany.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.

Valentini, G., Hamann, H., and Dorigo, M. (2015). Efficient decision-making in a self-organizing robot swarm: On the speed versus accuracy trade-off. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 1305–1314. IFAAMAS, Richland, SC.

Vassilev, V. K., Fogarty, T. C., and Miller, J. F. (2000). Information characteristics and the structure of landscapes. *Evolutionary Computation*, 8(1):31–60.

Watson, R. A., Ficici, S., and Pollack, J. (1999). Embodied evolution: Embodying an evolutionary algorithm in a population of robots. In *IEEE Congress on Evolutionary Computation*, pages 335–342. IEEE Press, Piscataway, NJ.

Watson, R. A., Ficici, S., and Pollack, J. (2002). Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18.

Whiteson, S., Stone, P., Stanley, K. O., Miikkulainen, R., and Kohl, N. (2005). Automatic feature selection via neuroevolution. In *7th Genetic and Evolutionary Computation Conference*, pages 1225–1232. ACM Press, New York, NY.

Whitley, L. (1991). Fundamental principles of deception in genetic search. In *1st Workshop on Foundations of Genetic Algorithms*, pages 221–241. Morgan Kaufmann, San Mateo, CA.

Wischmann, S., Stamm, K., and Wörgötter, F. (2007). Embodied evolution and learning: The neglected timing of maturation. In *9th European Conference on Artificial Life*, pages 284–293. Springer, Berlin, Germany.

Wolpert, D. H. and Macready, W. G. (2005). Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation,*, 9(6):721–735.

Yamauchi, B. and Beer, R. (1994). Integrating reactive, sequential and learning behavior using dynamical neural networks. In *3rd International Conference on Simulation of Adaptive Behavior*, pages 382–391. MIT Press, Cambridge, MA.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.

Yosinski, J., Clune, J., Hidalgo, D., Nguyen, S., Zagal, J., and Lipson, H. (2011). Evolving robot gaits in hardware: the hyperneat generative encoding vs. parameter optimization. In *20th European Conference on Artificial Life*, pages 890–897. MIT Press, Cambridge, MA.

Yuan, B. and Gallagher, M. (2007). Combining meta-EAs and racing for difficult EA parameter tuning tasks. In *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, chapter 6, pages 121–142. Springer, Berlin, Germany.